EXACT AND APPROXIMATE ALGORITHMS FOR FLOW SHOP SCHEDULING

by

T. V. L. N. SIVAKUMAR

DEPARTMENT OF INDUSTRIAL AND MANAGEMENT ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY KANPUR

JANUARY, 1995

EXACT AND APPROXIMATE ALGORITHMS FOR FLOW SHOP SCHEDULING

A *Thesis Submitted*
in Partial Fulfillment of the Requirements
for the Degree of
Doctor of Philosophy

by
T.V.L.N. Sivakumar

to the
DEPARTMENT OF INDUSTRIAL AND MANAGEMENT ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY KANPUR

January 1995

CERTIFICATE

It is certified that the work contained in the thesis entitled **"EXACT AND APPROXIMATE ALGORITHMS FOR FLOW SHOP SCHEDULING"**, by **"T.V.L.N. Sivakumar"**, has been carried out under my supervision and that this work has not been submitted elsewhere for a degree.

[illegible]

S. Sadagopan
Department of Industrial and Management Engineering
Indian Institute of Technology
Kanpur

January 1995

IME-1995-D-SIV-EXP

SYNOPSIS

Scheduling can be viewed as an efficient way of allocating limited resources to the tasks vying for them. Depending on the configuration of resources, the precedence constraints on tasks, the number of processors and the objective function to be optimized, scheduling problems can be classified into different categories. Some well known categories are job shop, flow shop and parallel machine shop. Flow shop, the focus of this thesis, can be defined as a shop with m machines and n jobs, in which each job j has exactly m operations and the precedence constraint on operations of each job is linear and equal to $(1, 2, \cdots, m)$. Flow shop is a special case of job shop in which all jobs need not have same machine order. A permutation flow shop is defined as a flow shop in which job order on each machine is same. Essentially, any scheduling problem can be considered as an optimization problem defined over a finite set of active schedules. Hence an optimal solution can be obtained by enumerating all the elements in this finite set. But, the cardinality of this set is so large that enumeration of the elements of this set is not practical but for the problems of very small size. Therefore more refined methods taking into account the specific structure of a given problem are required to solve the problem efficiently. These methods fall into two categories: exact methods which guarantee optimal solutions and approximate methods which generate near optimal solutions.

With the help of combinatorial analysis it is sometimes possible to design efficient algorithms that obtain optimal solutions within a predictable amount of computational time which increases at most polynomially with the size of the input problem. On the other hand, it seems necessary to employ implicit enumeration methods whose running time is less predictable and in the worst case may increase exponentially with

size of the input problem. Results in complexity theory show that such methods may be unavoidable since there exists a class of problems with a property that if there exists an efficient method to solve a problem belonging to this class then all other problems in this class will have an efficient method for solving them. This class of problems is known as NP complete problems. One implication of the above results is that it is highly unlikely that efficient exact algorithms can be found for the NP complete problems. So one advantage of showing that the problem of interest belongs to this class is that more effort can be devoted for designing better implicit enumeration techniques or efficient approximate methods. Most of the scheduling problems including flow shop scheduling problem belong to this class. Hence efficient exact methods are available for only a few restricted (special) cases of the general problem.

Most of the practical problems are of significant size and do not necessarily obey the special case restrictions. Implicit enumeration techniques with their worst case exponential behavior are useful only for solving problems of small size or for solving the problems where computational time is not a bottleneck or where obtaining exact solution is unavoidable. In most of the practical situations a scheduling problem can be considered to be effectively solved even if a near optimal solution is obtained. In these types of situations approximate algorithms are used for obtaining good approximate solutions in reasonable time. Approximate algorithms can be divided into two categories: problem specific heuristic and general heuristic methods. First type of heuristics are designed with the help of insight gained into the problem while doing combinatorial analysis or while designing polynomial time algorithms for special cases of the problem. Since the basic principles on which these heuristics are built are dependent on the underlying problem, these heuristics are restricted in their scope and cannot be modified to solve different types of problems. The recent trend in designing heuristics is to use a search method based on some natural phenomena. The principles used in this type of heuristics are general and they can be adapted to solve different types of problems.

Two different types of flow shops are treated in this thesis. The first type is a two machine flow shop problem with mean flow time as optimization criterion and the second is a two machine flow shop problem with release times and with makespan as optimization criterion. Since the above two problems are proven to be NP hard, polynomial time exact algorithms are developed only for their special cases in second chapter.

Let $t_i[0], t_i[1]$ and $t_i[2]$ denote release time, processing requirement on machine one, and processing requirement on machine two respectively for job i. Then the exact algorithms developed for the special cases of above mentioned problems are as follows:

- Two machine flow shop problem with zero ready times (i.e., $t_i[0] = K \ \forall i$) and with mean flow time as optimization criterion.

  1. When $\min(t_i[1]) \geq \max(t_i[2])$, the SPT ordering on first processor gives optimal sequence.
  2. Even when we reverse the above constraint i.e. $\min(t_i[2]) \geq \max(t_i[1])$, the problem remains an easy problem to solve and it can be solved by repeated (n times) applications of SPT rule on second machine processing time.
  3. When $t_i[1] \geq t_i[2] \ \forall i$, then the SPT ordering on first processor gives optimal sequences.
  4. When we reverse the above constraint i.e. $t_i[2] \geq t_i[1] \ \forall i$, unlike the above mentioned reversed problem, the problem no longer remains an easy problem to solve, and becomes NP hard. A proof is given to that effect.

- Two machine flow shop problem with release times and with makespan as optimization criteria.

1. $\min(t_i[1]) \geq \max(t_i[0])$; repeated applications of Johnson's rule.
2. $\min(t_i[1]) \geq \max(t_i[2])$; repeated applications of Early Release Time rule.
3. $t_i[2] \geq t_i[1]\ \forall i$; applications of dynamic Johnson's rule.
4. $t_i[1] \geq t_i[2]\ \forall i$; then the problem is proven to be NP hard.

In Third chapter we present effects of special cases on general branch and bound algorithms. There exist some dominance rules developed for 3 machine flow shop problem without release times. We have developed dominance rules based on special cases for the same problem and compared them with those reported in literature. Our study shows that none of these dominance rules are effective in reducing the computational time of $B\&B$ if used for solving random problems. Though one of the earlier reported dominance rules curtails the search by $B\&B$ to a significant extent, the cost for checking the related dominance conditions is so high that it outweighs the time advantage gained by curtailing the search and thus results in deteriorated time performance of the $B\&B$. We have conducted a study on the biased problems (they are called biased because certain percentage of jobs are made to obey special case conditions) to find the minimum percentage of jobs which obey special case conditions, at which using these dominance rules becomes advantageous. This study showed that if the job set contains more than 50% of the jobs that obey special case constraints then there can be significant savings in the search and in the CPU time. We have conducted a similar study for two machine flow shop problem with release times and with makespan as optimization criterion. Theoretical and computational results regarding correct use and effectiveness of dominance rules are presented.

Approximate algorithms (problem specific and general) are developed for the flow shop problem with release times. This problem has received very little attention from researchers in spite of the generality of the problem in comparison with the flow shop problem without release times. In Chapter four we have proposed two problem specific heuristics, an improvement method based on decomposition principle and a hybrid general search algorithm which combines the principles of both simulated annealing and tabu search. Decomposition improvement method was compared with local search improvement methods in pairwise exchange neighborhood for different initial solutions. The problem specific heuristics were compared with other problem specific heuristic reported in literature. To study how the problem specific heuristics and hybrid algorithm compare with general purpose heuristics, we have implemented simulated annealing and tabu search methods. Parametric study was carried out to find best parameters for decomposition improvement method, simulated annealing and tabu search. We have adapted a good problem specific heuristic originally designed for flow shop problem without release times to solve the problem of our interest to see how well this problem specific heuristic compares with other heuristics. One of the objectives of this study is to compare problem specific heuristics with known worst case performance bounds, with general purpose heuristics. Such a study can be conducted for problems having only two machines because for problems having more number of machines, there exist no approximate algorithms with performance guarantees. Because of the above reason, we have restricted our study to the problems having only two machines.

The major conclusions reached through our study are as follows. The two heuristics proposed by us outperform other problem specific heuristics. Decomposition exchange method, at the expense of marginal increase in CPU time, outperforms local search method which uses pairwise exchange neighborhood, for almost all initial solutions. For the improvement methods we have tested, there seems to be a matching heuristic to get the initial solution. General purpose heuristics outperform problem specific heuristics at the expense of high CPU time. The heuristic which works well for flow shop problem without release times does not work well when modified to solve our problem of interest. Our hybrid algorithm outperforms both tabu search and simulated annealing. Almost all heuristics tested give good quality solutions i.e. average deviation from optimal solution is less than 10%. Based on the results obtained for various types of heuristics, a meaningful insight into the problem was gained.

Dedication

To my mother.

Acknowledgements

It is my pleasure to acknowledge the constant effort put in by my supervisor, Dr. S. Sadagopan, in bringing the thesis to this meaningful form. Without his constant encouragement and criticism this work would have been impossible. The tips, regarding the ways of research, given by him are worth remembering through out my research career. Thanks are due to Mrs. Sadagopan for not so infrequent light meals which have kept my home sickness at a bay.

I would like to thank Dr. Ravi Ahuja for introducing me to Network Flows and thus broadening my knowledge and the areas of interest. His suggestions during final stages of my stay here were extremely useful in molding my goals. I thank Dr. Kripa Shanker for his encouragement and unforgettable dinners at his home. I would like to thank, Dr. Mittal, Dr. Batra, Dr. Bagchi, Dr. AP Sinha, Dr. RRK Sharma and Dr. Rahul Varman for making my stay at IITK a meaningful experience. I thank our scientific officers, Mr. Arora and Mr. Mohan for their full cooperation during my thesis and their enthusiastic greetings every day morning. I thank our office staff, Mr. JK Mishra, Mr. GK Mishra, Mr. IP Singh, Mr. Bahadur Singh, Mr. Amrit Lal, Mr. Ayodhya Prasad and last but not the least Mr. Budhi Ram for their ever willing helping hands.

I would not like to acknowledge the patient proof reading by Neeraj, which has tested, probed and extended the tiny boundaries of my patience, just to spite him. I do not envy his taste in literature, music and his penchant for nuances in arguments and I do not wish I had acquired them if not fully at least partially. Through him I met Dr. Sanjay who etched indelible clichs in my brain so that I can use them if ever I become a *Pseud.* The illuminating discussions, *sermons on the mount* in our parlance, with Kutub are something to be pondered on when one is left with nothing but despondency (the only thing I have in plenty) to live with. Think of the occasional Arabic he garnishes his monologues with, you have every thing that is incomprehensible which got to be comprehended to make the life more comprehensive. There is no dull moment in the ambiance when Kutub is around. Dinners and sweet dishes at his home are the ones that wander my memory lane even long after the world is forgotten and become irrelevant to me.

I thank Ajay for his constant help in procuring papers from US which are mysteriously invisible here even though they leave the traces of their origins. I thank him in advance for what he would do from now on. Anu cannot be forgotten for his famous dialogue, "*Chalo be, ... hai*" and for the papers he sent from Canada. I would like to thank Sreekant, all the three, B, S and T, and Rahul singh for their help. I thank Jaganmohan Reddy, Shanker for helping me in the lab work.

Acknowledgements would be incomplete if I do not mention Balaya and his family (old and new), Ramki and his relatives (he might not like me for mentioning his appendages, but who cares?) for the extremely happy moments they have inserted in my life. Pikku's dinners (mostly given at a future date) are something to write home and can quench any hunger with their gargantuan menus.

I thank Vijay Sai, Suresh Babu, Nasy, Ravindra, Bhaskar Shyam, Saya, Subba Rao (70 mm), Reddy, Jagirdar, Sunil Rajotia, Rajeeve Gupta, GK, Kak, Phatta, Shanker, Tapo, Khanna and Pandeji for making my stay comfortable here.

Finally I would like to thank all second and first year M. Tech students for giving me unstinted use of the resources. Special mention is to Vijay and Yashkant for typing some part of the thesis and preparing some of the diagrams used in this thesis.

January, 1995 (T.V.L.N. Sivakumar)

Contents

List of Tables

List of Figures

Chapter 1

INTRODUCTION AND LITERATURE SURVEY

Scheduling can be regarded as an efficient way of allocating limited resources to the tasks vying for them. The 'efficient way of allocating' can be viewed in many different ways. It can be viewed as maximization of resources, maximizing throughput, minimizing work-in-process inventory, minimizing mean tardiness of jobs or in any other way that is beneficial to the concerned organization. In many of the practical situations, the available resources are limited and are sought after by many tasks. Hence, an efficient scheduling procedure can be an invaluable tool for effective utilization of resources. Some examples of the need for scheduling activity in real life are: processor scheduling in parallel computing, machine scheduling in shop floor, fleet scheduling in transportation industry etc. Scheduling has attracted the attention of many researchers from different areas not only because of its wide applicability but also because of its problem structure which is of interest to many working in the area of combinatorial optimization.

1.1 Scheduling Problem

One of the formal descriptions of scheduling problem is as follows [99]. Suppose that we have a number of jobs each of which consists of a given sequence of operations which are performed on a number of machines. To perform a job, each of its operations must be processed in the order given by the sequence. The processing of an operation requires the use of a particular machine for a given duration, the processing time of the operation. Each machine can process only one operation at a time. Given a cost function (optimization criterion) by which the cost of each feasible solution can be measured, we want to find a processing order on each machine such that the corresponding cost is minimized. The terms 'Machines' and 'jobs' in this description can stand not only for machines and jobs in a shop floor but also for various other things depending upon the context, some examples are: hospital equipment and patients, teachers and classes, dockyards and ships etc.

Based on the configuration, availability and capability of resources and the technological constraints on the task set, the scheduling problems can be classified into many classes of problems. Most of these restrictions on resources are stated as assumptions (restrictions) while dealing with a problem. To decide whether a given problem can be modelled as a scheduling problem, the explicit and implicit restrictions imposed on the problem are to be carefully considered. The restrictions along with the optimization criteria lead to the detailed classification of scheduling problems.

1.1.1 Assumptions

The assumption that are normally stated in scheduling literature are [99]

J1 The set of jobs, J, is known and fixed.

J2 All jobs are available at the same instant and are independent.

J3 All jobs remain available during an unlimited period.

J4 At a given time, each job can be in any one of three states: waiting for the next machine, being operated by a machine or having passed its final machine.

J5 All jobs are equally important.

J6 Each job is processed by all the machines assigned to it.

J7 Each job is processed by one machine at a time.

M1 The set of all machines, M, is known and fixed.

M2 All machines are available at the same instant and independent.

M3 All machines remain available during an unlimited period.

M4 At a given time, each machine can be in any one of the three states: waiting for the next job, operating on a job or having finished its final job.

M5 All machines are equally important.

M6 Each machine processes all the jobs assigned to it.

M7 Each machine processes one job at a time.

JM1 All processing times are fixed and are sequence-independent.

JM2 Each operation once started must be completed without interruption.

JM3 The processing order per job is known and fixed.

JM4 The processing order per machine is unknown and has to be fixed.

1.1.2 Optimization Criteria

The optimization criteria that are well studied in the literature are regular measures whose value with respect to a given schedule is completely determined by the job completion times. Though this seems to be a serious limitation, in practice most of the costs that can be controlled by a scheduling system can be represented using regular measures [10, 99]. These criteria can be classified into two groups:
1. Due-date based criteria and
2. Non due-date based criteria.
The former criteria are mainly used when there are due-dates (dead lines to deliver jobs) associated with each job and when failure to meet a dead line results in measurable loss to the system. The second type of criteria are mainly used either in absence of due-dates or when meeting the due-dates is not important. Here we list a few of both types of criteria:

- Due-date based criteria:

  1. Minimization of maximum lateness.

     min $Z = L_{max}$

     L_{max} = maximum lateness = $\max_i(L_i)$

     L_i = lateness of the i^{th} job = $c_i - d_i$

     c_i = completion time of i_{th} job

     d_i = due date of i_{th} job

  2. Total tardiness, sum of weighted tardiness and number of tardy jobs where tardiness is defined as positive lateness i.e., $\max(0, L_i)$

- Non due-date based criteria:

1. Minimizing maximum completion time. This criterion is also known by other names like makespan and schedule length.
   $C_{max} = \max\{C_i | i = 1 \cdots n\}$
2. Minimizing the sum of completion times, $\sum_{i=1}^{n} C_i$
3. Minimizing weighted sum of completion times $\sum_{i=1}^{n} W_i \times C_i$, where W_i is the weightage assigned to the ith job.

1.1.3 Classification

It can be seen that the assumptions J1 and M1 distinguish static scheduling problems from dynamic scheduling problems. Relaxation of assumption J2 leads to a class of scheduling problems known as scheduling problems with job release times and/or precedence constraints. Incorporating due dates into the scheduling model violates assumption J3. Thus by relaxing some of the above mentioned assumptions we get different classes of scheduling problems. A detailed discussion on classification of scheduling problems is available in The book 'Theory of scheduling' by Rinnooy Kan [99].

We define the flow shop scheduling which is the focus of this thesis.

1.1.3.1 Flow Shop Scheduling

Flow shop, the focus of this thesis, can be defined as a shop with m machines and n jobs, in which each job j has exactly m operations and the precedence constraint on operations of each job is linear and equal to $(1, 2, \cdots, m)$. Flow shop is a special case of job shop in which all jobs need not have same machine order. A permutation flow shop is defined as a flow shop in which job order on each machine is same. Flow shop scheduling problem is to find the start and finish times of the operations of jobs on each machine such that a chosen criteria is optimized

The following assumptions are made regarding flow shops for the rest of the thesis.
Assumptions:

- A set of n multiple operation jobs are available for processing.
- Each job requires m operations and each operation requires a different machine
- Set up times for operations are sequence independent and are included in processing times.
- m different machines are continuously available.
- Individual operations are not pre-emptable.

1.2 Solution Methods

Essentially each scheduling problem is an optimization problem defined on the finite set of active schedules. Hence an optimal solution can be obtained by enumerating all the elements in this finite set. But the cardinality of this set is so large that complete enumeration of the elements of this set is not practical other than for the problems of very small size. Therefore more refined methods taking into account the specific structure of a given problem are required to solve the problems efficiently. With the help of combinatorial analysis it is sometimes possible to design efficient algorithms that obtain optimal solutions within a predictable amount of running time which increases at most polynomially with the size of the input problem. On the other hand, it seems necessary to employ implicit enumerative methods whose running time is less predictable and in the worst case may increase exponentially with the size of the problem.

The results in the complexity theory shows that there exists a class of difficult combinatorial problems with the property that an efficient algorithm for any one of these problems would provide efficient algorithms for all other problems belonging to this class. This class of problems are known as NP hard problems (more precisely, their decision versions are known as NP complete problems). Since this set contains such well researched problems as the traveling salesman problem, 0–1 integer programming problem etc., it is very unlikely that efficient solution procedures can be found for these problems. The advantage of showing a scheduling problem to belong to this class is that more effort can be put in designing either better implicit enumeration techniques or in designing efficient approximate algorithms.

Since most of the scheduling problems belong to this class of problems [27, 44], it seems inevitable to use either implicit enumeration methods or approximate algorithms for solving these problems. In most of the practical situations a scheduling problem can be considered to be effectively solved even if a near optimal solution is obtained for the problem since the search for an optimal solution is too time consuming. Hence the role of approximate algorithms to quickly get good approximate solutions cannot be ignored.

The solution methods to solve scheduling problem can be divided into two categories:

1. The solution methods to obtain optimal solutions (exact algorithms) and
2. The solution methods to quickly obtain good suboptimal solutions (approximate algorithms).

1.2.1 Exact Algorithms

As mentioned above, these can be divided into two different categories based on their worst case running times in terms of input problem size:

1. Polynomial time algorithms and
2. Exponential time algorithms

Polynomial time algorithms were, till now, found only for problems of small size or for some special cases of scheduling problems. Though these methods have limited scope, they can give good insight regarding general scheduling problems and thus help in designing good heuristic methods. Exponential algorithms are used for solving problems for which polynomial time algorithms could not be found. One of the very frequently used exponential algorithms in scheduling is branch and bound algorithm. Bulk of the research in designing these the branch and bound algorithms is concerned with designing tighter lower bounds, better branching rules and more efficient ways of pruning search branches using conditions like dominance rules. Though these exponential algorithms can be used for solving problems of any size, usefulness of these methods, owing to their time complexity, is limited to problems of small size, or to situations where finding an optimal solution is unavoidable.

1.2.2 Approximate Algorithms

Due to inherent intractability of scheduling problems, it is necessary to use approximate algorithms to solve most of the practical problems in order to obtain good solutions in reasonable time. Traditionally, heuristics are designed with the help of insight gained into the problem while doing combinatorial analysis or while designing polynomial time algorithms. The recent trend in designing heuristics is to employ some intelligent search method (like, simulated annealing, tabu search and genetic algorithms etc.). Some attempts have also been made at developing real-time algorithms and adaptive heuristics which can consider the dynamic nature of system status [25, 65, 66, 80, 101, 125].

1.3 Literature Survey

Scheduling literature is vast and to present a comprehensive literature survey of all the problems that are considered in the literature is beyond the scope of this work. Since the subject matter of this thesis is flow shop scheduling, we restrict attention to only flow shop scheduling for the remainder of this chapter with occasional digression towards job shop scheduling whenever necessary.

1.3.1 Notation

The following notation is followed for the rest of the chapter.
Let
$t_i[0]$ be the release time of the job i,
$t_i[j]$ be the processing time of the job i on the machine j for $1 \leq j \leq m$

Interest in flow shop scheduling research started with the publication of a paper by S. M. Johnson [67] in 1954. It establishes some properties of flow shop and gives polynomial time algorithms for solving restricted flow shops (two machine flow shop and special cases of three machine flow shop problem) for makespan optimization criterion. Later it was established that general flow shop scheduling problem for every interesting optimization criterion is NP hard [27, 44, 99]. However, researchers were able to design some polynomial time algorithms which can solve some restricted versions of the general problem. Given the restricted scope of these polynomial time algorithms, it is not surprising to find that considerable amount of research effort is spent in designing good approximate algorithms for obtaining near optimal solutions.

Further literature survey is presented according to the type of solution method i.e. exact and approximate methods.

1.3.2 Exact Solution Methods

This subsection is divided into two parts. First part discusses the problems which are solvable in polynomial time. Second part discusses controlled enumeration techniques which, in the worst case, run in exponential time for solving flow shop scheduling problems.

1.3.2.1 Polynomial Time Algorithms

Johnson [67] gave an algorithm to solve the two machine flow shop problem with makespan optimization criterion. In the same paper, he gave an algorithm to solve two special cases of the three machine flow shop problem. The conditions he imposed on operations of jobs so as to arrive at special cases are:

1. $\min(t_i[1]) \geq \max(t_i[2])$, or
2. $\min(t_i[3]) \geq \max(t_i[2])$.

For both the cases, a virtual two machine flow shop is created with modified processing times for two stages as $t'_i[1] = t_i[1] + t_i[2]$ and $t'_i[2] = t_i[2] + t_i[3]$. The virtual two machine problem is solved by Johnson's algorithm given for the two machine flow shop problem to obtain optimal sequence for the original problem. Cheng [22] presents a faster implementation of the Johnson's rule to sequence two machine flow shop problem with makespan criterion. This implimentation can also generate all possible optimal sequences obtainable from Johnson's rule.

Arthanari and Mukhopadhyay [5] reported two special cases of three machine flow shop problem with makespan optimization criterion:

1. $\min(t_[i]2) \geq \max(t_i[1])$

The optimality rule for a given first job is: the job i will precede job j if $\min(t_i[2], t_j[3]) \leq \min(t_i[3], t_j[2])$.

2. $\min(t_i[2]) \geq \max(t_i[3])$

The optimality rule for a given last job is: the job i will precede job j if $\min(t_i[1], t_j[2]) \leq \min(t_j[1], t_i[2])$.
For both these cases, n sequences are generated with each job as first (or last) job and the best out of these n sequences is an optimal sequence.

Burns and Roorker [17] prove that when the second stage of a three machine problem is recessive i.e., $t_i[2] \leq \min(t_i[1], t_i[3])\ \forall i$, then the problem is equivalent to the two machine problem with modified processing times as given by Johnson [67] above. They also reported [16] that when Johnson's two stage rule applied to first two stages yields the same sequence as when applied to last two stages as well as first and third then the sequence thus obtained is optimal. Through separate work [18], they give an algorithm to solve three machine flow shop problem satisfying the condition $[\min(t_i[1], t_j[2]) - \min(t_j[1], t_i[2])] \times [\min(t_i[2], t_j[3]) - \min(t_j[2], t_i[3])] \geq 0\ \forall\, i, j\ \in\ I$

Szwarc [115] proves that for any three machine flow shop problem with constant second stage i.e., $t_i[2] = t_{i+1}[2] \forall\, i$, if natural sequence $\langle 1, \ldots, n\rangle$ yields $t_1[1] \leq t_2[1] \leq \cdots \leq t_n[1]$ and $t_1[3] \geq t_2[3] \geq \cdots \geq t_n[3]$ then the permutation $[1, \ldots, n]$ is optimal sequence.

Smith et al [106] identify a class of flow shops as ordered flow shops and define them as flow shops in which

1. if a particular job has smaller processing time on any machine than any other job, then the former will have all the processing times of operations less than all other jobs on all machines, and
2. the machine with minimum processing time for an operation of a given job will also have the minimum processing time for every other job.

They also define a special ordered flow shop as an ordered flow shop in which greatest processing times occur either on first machine or the last machine. The algorithm presented by them for solving special ordered flow shops optimally is:

Case 1 Last machine has the greatest processing time—sequence with jobs in ascending order on any specific machine of their processing times is optimal.

Case 2 First machine has the greatest processing time—sequence with jobs in descending order of their processing times on any specific is optimal.

Case 3 In case of tie between first and last machine, both the sequences are optimal.

Chin and Tsai [23] presented an $O(n^7)$ algorithm to find an optimal sequence for a three machine flow shop problem when $t_i[1] \geq t_i[2] \geq t_i[3]$ or $t_i[1] \leq t_i[2] \leq t_i[3] \; \forall \; i$.

Smith and Baker [107] have studied the occurrences of special cases in three machine flow shops. The special cases which they have considered for their study are the following:

- C1: M_1 dominates M_2.
- C2: M_3 dominates M_2.
- C3: M_2 dominates M_1.
- C4: M_2 dominates M_3.
- C5: Recessive second stage.
- C6: Johnson's conjecture [16].
- C7: Lower bound criterion: If the makespan corresponding to an optimal sequence to the pseudo problem of Johnson's approximate method equals the make span for the same sequence of three machine problem $+\sum t_i[2]$, then the sequence is optimal.
- C8: Constant second stage.

Various experimental setups they have considered for their experimentation are:

- S1: Random: Processing times are independent and are uniformly distributed.
- S2: Ordered flow shop.
- S3: Constant second stage.
- S4: Correlated Shop: If the processing of a job is large on any one particular machine then the jobs processing times on other machines also tend to be large.
- S5: Trend shop: The processing times are positively correlated with machine number.
- S6: Correlated trend shop: Combination of S4 and S5.

After the experimentation they have arrived at the following conclusions:

- Of all the conditions tested the most successful is C7.
- Half of the total problems tested contained at least one special case.
- S4 and S6 have fewer successes as the problem size increased while the opposite is true for S5.
- In S1, S4, S5 and S6, C7 is the only successful rule.
- Unless special shop structure is involved (for example, ordered flow shop or constant second stage) the other conditions are virtually ineffective.
- Correlation among processing times presents greatest difficulty in search for an optimal sequence, particularly as the problem size increases.

It is surprising to note that all the special cases developed are for makespan riterion. To the best of our knowledge there is no published literature about special ases of flow shops with criteria other than makespan.

.3.2.2 Exponential Algorithms

t is obvious from the above discussion that some special cases of flow shop scheduling problems are amenable to solutions by polynomially bounded algorithms. But here are many, in fact most, flow shop scheduling problems that belong to a class ›f problems known as NP-hard problems — a class for which polynomially bounded ılgorithms are not likely to be found. Since most of these problems can be formulated ıs discrete optimization problems (for example, see [41]), a simple but prohibitively time consuming total enumeration method can be employed to find optimal solution. Rardin and Parker [90] write that although the total enumeration is a hopeless strategy for most discrete optimization problems, it does not follow that the notion of enumeration should be completely rejected. As an alternative to total enumeration, we can employ controlled/partial enumeration to obtain a desired solution. Partial enumeration techniques rely on enumerating solutions in a controlled manner. They divide feasible region into sub regions and fathom them explicitly or implicitly based on some mathematically conservative estimates (e.g. lower bound) of optimal solutions. This is the fundamental principle underlying partial enumeration techniques.

In flow shop scheduling context partial enumeration techniques fall into two categories:

1. Branch and bound methods, and
2. dominance rules.

In the worst case, both these methods search all the solution space. Thus in pure complexity theoretic sense, neither is an improvement over total enumeration. These methods are normally employed when obtaining an optimal solution is unavoidable ɪr when the time taken to obtain a solution is not a primary concern to the decision naker.

3ranch and Bound Method

In order to use the branch and bound algorithm, the problem should be represented ɪs a tree where each node represents a partial solution. In addition, for each node, a lower bound on the objective function for all the nodes emanating from it must be written. Most of the research in this area is concerned with finding tight lower bounds so that the amount of enumeration can be relatively minimized. For makespan criterion, three different kinds of lower bounding techniques are employed.

1. **Machine Based Bounds:** In machine based bound calculation, each machine is treated as a bottleneck machine and all other machines are clubbed and treated as a single non-bottleneck machine. Solution to the above problem is considered as a lower bound. Ignall and Schrage [64] and Lomnicki [75] have developed these bounds independently. Later these were improved upon by McMohan and Burton [77].

2. **Job Based Bounds:** For a given partial sequence, each job is considered as the next job and rest of the unscheduled jobs are assumed to cause no idle time, and the estimated schedule length is treated as lower bound. The best of the bounds thus obtained is treated as final lower bound.

3. **Composite Bound:** The bounds are calculated using above two methods and best of the above is considered to be final lower bound.

Lagweg [74] developed a theory based on bottleneck and non-bottleneck machines and derived all the above bounds in addition to two new bounds. Ashour [7] developed non interference bounds. Since only one job can occupy a machine at a time conflicts ıust be resolved when two or more jobs are ready to be processed simultaneously t a given machine. The resolution of such a conflicts on machine m gives rise to a oninterference bound. Baker [11] in his study on the exact algorithms for makespan riterion concluded that composite bounds are better and that machine based bounds .re suitable only for small job sets and uncorrelated shops.

Ignall and Schrage [64] developed machine based bounds for total flow time crite-ion for a two machine flow shop problem. Later these were extended by Bansal [12] 'or m×n flow shop. Ahmadi [3] has improved these bounds by:

— formulating the problem as a discrete optimization problem,

— reducing discrete optimization problem to a single machine scheduling problem with release times and due dates, by series of approximations, and

— solving it by SRPT (Shortest Remaining Processing Time) heuristic.

Townsend [118] considered n jobs m machine problem to minimize maximum tardiness. C. N. Potts [93] considered minimization of number of tardy jobs in a m×n flow shop. He uses single machine relaxation for calculating the bounds. Daniel [31] considered a bi-criteria flow shop scheduling problem with makespan and maximum tardiness as optimization criteria. For obtaining bounds on partial schedules, he uses the heuristic developed by Nawaz et al [81] for makespan and EDD ordering on bottleneck machine for maximum tardiness. He considers each machine as a bottle neck machine and calculates lower bounds. Best of the lower bounds is taken as the final lower bound. Uskup and Smith [120] consider an n job sequencing problem in a two stage production environment, each stage being equipped with a single facility. Jobs are to be sequenced without violating due dates so as to minimize total sequence dependent set up time. The algorithm proposed by them employs controlled enumeration through branch and bound technique. The underlying principle of their algorithm is that controlled enumeration is used over the schedules which satisfy the deadlines. For a formal description of the algorithm refer to the original work.

Dominance Rules

The dominance rules are also known as elimination methods in literature. The basic principle used in this type of solution method is that for a given partial sequence, two jobs i and j are checked to see whether one of them dominates other. If the dominance is established, then the dominated sequences are eliminated from final consideration. Similar procedure is repeated with further partial sequences until all the remaining partial sequences are undominated which are explicitly enumerated for an optimal sequences.

Dudek and Teuton [36] were the first ones to report dominance conditions through combinatorial analysis. Their algorithm begins with a partial sequence σ, which is a set of already scheduled jobs. Two jobs, i and j are compared by $(m - 1)$ conditions. If all the conditions are met, then job j is eliminated from consideration. This algorithm was originally thought to be a general one to get an optimal sequence but a counterexample formulated by Karush [68] proved otherwise. Later Smith and Dudek [105] eliminated the drawbacks of this algorithm and also gave a proof of sequence optimality.

For the three machine flow shop problem, Gupta and Reddy [56] developed new dominance conditions which eliminated an important drawback of the existing dominance conditions [55, 78, 112] that the jobs should have their least processing time on either first or third machine. Szwarc in [112] proved that the elimination methods developed by him in [111] are optimal and outlined a method to construct a general counterexample to any procedure that claims to remove more sequences than this optimal method. He also gave a counterexample to show that elimination methods given by Bagga and Chakravarti [9] eliminate all optimal sequences from consideration and thus result in suboptimal solutions. Through another work [113], Szwarc examines the mathematical structure of a dominance condition given by Gupta and Reddi [56] for the three machine flow shop problem. He showed that the condition developed in [56] may eliminate fewer sequences than the conditions developed in [55, 78, 112]. Then he gives a sequential procedure to combine the two conditions in order to achieve better results.

Baker [11] by his experimental study concluded that the algorithms based on dominance criteria are inferior to algorithms based on branch and bound methods. He suggests use of a composite approach which uses branch and bound technique on the dominant set of sequences.

1.3.3 Approximate Algorithms

The approximate algorithms (heuristics) for flow shop scheduling can be divided into two categories:

1. problem specific heuristics, and
2. general purpose heuristics.

Problem specific heuristics are the heuristics which use problem specific knowledge for solving the problem. The scope of these heuristic methods is narrow i.e., these heuristics cannot be used for solving different types of problems. In contrast, general heuristic methods employ general search methods which, in general, do not utilize any specific knowledge about the problem they are to solve. Some of the notable general heuristics are simulated annealing, tabu search, genetic algorithms and filtered beam search. Nevertheless, the knowledge of the problem to be solved may some times be utilized to increase the quality of the solutions obtained and/or to speed up the heuristic. Example are: Crossover operators defined for TSP in genetic algorithms, and filtering and beaming heuristics used in filtered beam search. Most of the general heuristic search methods use a analogies form other sciences to guide the search process. This aspect of modern heuristics is evident in the way GAs mimic evolutionary process, or simulated annealing mimics physical annealing process. It is observed that general heuristic methods outperform problem specific heuristics (see chapter four).

A comprehensive treatment on design and analysis of heuristic methods is given by Judea Pearl [91]. The book, 'Modern heuristics techniques for combinatorial optimization problems', by Colin R. Reeves [96] details the modern heuristic techniques. Now we look into the two categories in more detail.

1.3.3.1 Problem Specific Heuristics

Page [88] gives four different types of heuristics for solving m machine flow shop scheduling problem for makespan criterion. One of these heuristics utilizes local search method in pair-wise exchange neighborhood. Palmer [89] suggests priority based sequencing and gives the following rule to assign priority indices to jobs: $S_j = (m-1)t_j[m] + (m-3)t_j[m-1] + \cdots - (m-3)t_j[2] - (m-1)t_j[1]$. After the jobs are assigned priorities, they are arranged in non increasing order of their priorities to get the final sequence. Gupta [57] suggests a different way of calculating priority indices for jobs:

$S_j = \frac{e_j}{\min_{1 \le k \le m-1}(t_j[k]+t_j[k+1])}$

where

$e_j = \begin{cases} 1 & if \quad t_j[1] < t_j[m] \\ -1 & if \quad t_j[1] \geq t_j[m] \end{cases}$ Then the jobs are arranged in nondecreasing order of their priorities.

The heuristic proposed by Ashour [6] is a truncated branch and bound algorithm i.e., the first feasible sequence generated is accepted as the solution to the problem. Campbell et al [19] proposed a heuristic based on multi stage application of Johnson's rule. The heuristic constructs $(m-1)$ sequences and chooses the best one in the following manner.

```
For i=1 to m

begin
```

```
        $t_j[1]' = \sum_{k=1}^{i} t_j[k]$ and $t_j[2]' = \sum_{k=1}^{i} t_j[m-k+1]$
        Apply Johnson's rule on aggregated two machine problem with $t_j[1]'$ and
        $t_j[2]'$ as processing times to get $i^{th}$ sequence.
end
```

Choose the best among m sequences.

This heuristic was analyzed by Nowicki and Smutnicki [82] for worst case performance bounds. The bound on the makespan of the sequence given by this heuristic is

$\frac{C^h_{max}}{C^{opt}_{max}} \leq \lceil m/2 \rceil$

And this bound is proven to be tight. Empirically, this heuristic is found to give near optimal results.

Dannenbring [32] proposed three heuristics for solving flow shop sequencing problem:

1. **Rapid access procedure**: Solve the Johnson's two machine problem with following weighted times. $t_j[1] = \sum_{i=1}^{m}(m-i+1) \times t_j[i]$ and $t_j[2] = \sum_{i=1}^{m} i \times t_j[i]$

2. **Rapid access with close order search**: Initial solution is obtained using the rapid access procedure and this solution is compared with its $(n-1)$ neighbors in the adjacent exchange neighborhood. The best solution is accepted as final solution.

3. **Rapid access with extensive search**: Initial solution is obtained by the application of the rapid access procedure and with this solution as starting point a local optimum is found in the adjacent exchange neighborhood.

Giglio and Wagner [47] formulated three machine flow shop scheduling problem as an integer programming problem. They solved it by ignoring integer constraints using simplex method. The solution thus obtained was rounded off to get integer solution to the original problem.

King and Spachis [70] proposed five different heuristics and through their empirical analysis found that one of their heuristics outperforms Campbell et al's heuristic [19]. These heuristics are constructive heuristics. They differ from one another in the way they try to obtain minimal *between jobs* delays. For formal description of these heuristics refer to the original work. Nawaz et al [81] proposed a heuristic based on greedy principle. It is computationally superior to Campbell et al's and Dannebring's [32] heuristics. This fact was verified by Turner and Booth [119]. Sarin and Lefoka [102] developed a heuristic based on a principle which is similar to the central principle of Nawaz et al's heuristic. The central idea of the heuristic is to sequence the jobs such that the idle time on last machine is minimal. Through their comparative results Sarin and Lefoka found that when the ratio of number of machines to number of jobs increases their heuristic outperforms the heuristic developed by Nawaz et al. In general, computational time for this heuristic is significantly smaller than that for Nawaz et al's heuristic. Taillard [116] reduced computational complexity of Nawaz et al's heuristic by using an efficient sequence construction and evaluation procedure.

Widmer and Hertz [124] transform flow shop scheduling problem into TSP using the notion of distance between jobs. The distance between two jobs, d_{ab}, is a measure of increase in the makespan if the job b follows job a. Insertion heuristic is used for finding the sequence and resulting sequence is improved upon by applying tabu search.

Taillard [116] compared improved heuristic of Nawaz et al with tabu search method for flow shop problem and found that tabu search outperforms the heuristic by Nawaz et al. Along with the performance results he has reported interesting statistics about the distribution of solutions with respect to their makespans.

Other optimization criteria, in comparison with makespan criterion, received very little attention. Gelders and Sambandam [46] presented four different types of heuristics for obtaining good solutions to flow shop sequencing problem when the sum of weighted flow time and weighted tardiness is to be minimized. The heuristics reported by them are based on the following principles:

1. Use dynamic dispatch rules, and
2. Give priority to the jobs which are most expensive to hold,
3. Fit the jobs in sequence such a way that idle time is minimal.

Jobs are sequenced according to the priorities assigned to them. Priorities are assigned based on the estimated tardiness of the job and the lower bound on completion time of batch for a given partial schedule. All the four heuristics vary only in the way these estimates are calculated.

Krone and Steiglitz [73] give a heuristic for solving flow shop problem when mean job completion time is optimization criterion. The basic principle on which this heuristic works is as follows:

Step 1. Start with a random sequence.

Step 2. Find the local optimum using local search method.

Step 3. Repeat steps 1 & 2 fixed number of times.

Step 4. Select the best sequence.

Step 5. Subject it to job passing (see the discussion on Nugents work in the above paper) to get a better non-permutation schedule.

Kohler and Steiglitz [71] presented exact, approximate and guaranteed accuracy algorithms for two stage flow shop problem with mean flow time as optimization criterion. Their approximate algorithms are based on non-backtracking branch and bound and on local search techniques. Their guaranteed accuracy algorithm employs a branch and bound algorithm with modified termination criteria. B&B procedure terminates whenever a feasible solution found meets the accuracy requirement. They also presented computational results comparing different heuristics.

1.3.3.2 General Purpose Heuristics

Recent trend in heuristic programming is to develop general heuristic methods which either mimic a natural process or use knowledge of the domain expert in guiding the search for a good solution. Many of these techniques originated in different areas and have been adapted to solve optimization problems successfully. The most potent technique for designing heuristics seems to be the combination of Artificial Intelligence and Operations Research [48]. For a comprehensive treatment of these heuristic methods, interested readers can refer to the book, 'Modern heuristic techniques for combinatorial optimization problems', edited by Colin R. Reeves [96]. Now, we describe some of the general purpose heuristics and their use in solving flow shop scheduling problem.

Simulated Annealing

This, basically, is a hill climbing algorithm which avoids local optima traps with decreasing probability in order to reach a global optimum. It tries to mimic physical annealing process where a hot material is cooled slowly so that the material settles down to a low thermal energy level. Its potential to solve combinatorial problems was first pointed out by Kirkpatrick et al [69]. This heuristic is used for solving many combinatorial optimization problems, including flow shop problem [86, 84, 85] and job shop scheduling problem [123, 121]. These specific implementations of simulated annealing vary mainly in the way the cooling schedules are implemented, the types of neighborhoods searched and the objective function defined. A review of the applications of simulated annealing can be found in [72, 122, 96]. Aarts and Laarhoven [96] reported convergence results of simulated annealing using homogeneous markov chains and Hajek [58] gives convergence results using non homogeneous markov chains. Usefulness of these convergence results in practical applications is limited, as they imply computational times which are exponential in problem size. Nevertheless, these convergence results can guide the decisions that are required to be made for implementing this algorithm. There is substantial amount of research work on statistical behavior of simulated annealing. Interested readers can refer to the books by Laarhoven and Aarts [122] and Aarts and Korst [1] for a detailed treatment of the technique. The necessary information that is required for implementing simulated annealing algorithm can be found in the chapter *Simulated Annealing* in the book edited by Colin R. Reeves [96] or in the papers [69, 37, 48].

The researchers [86, 84, 85, 124] who implemented simulated annealing for solving flow shop sequencing problem reported that this heuristic outperforms the best problem specific heuristic available [81]. This experimentation was carried out only for makespan criterion and there is no published attempt at using this technique for flow shop problems with other criteria. For a detailed description of algorithm and the parameter selection, refer to chapter four.

Tabu Search

Tabu search also is a variation of hill climbing algorithm which is designed to avoid local traps [51]. This heuristic approach is generally useful for solving problems of the following type:

Minimize $C(x)$

Subject to

$x \in X$

Tabu search may also be conveniently seen as a variation of neighborhood search where neighborhood is modified to reflect the current status of the process of the heuristic. It was first proposed by Glover [50, 49]. Seminal ideas of this heuristic were also developed by Hansen [59] in steepest ascent and mildest descent formulation. In contrast to simulated annealing which employs randomization as a means to avoid local optima traps, this method de-emphasizes randomization and uses history of the progress made by heuristic as a means of avoiding local optima traps. However, randomization is employed in a constrained way and is assigned the role of facilitating operations that are otherwise cumbersome to implement or whose strategic implementations are not clear [51]. Exception to the above principle occurs for the variant of tabu search called Probabilistic tabu search. A discussion on convergence of probabilistic tabu search was given by Faigle and Kern [38]. Tabu search is being successfully used for solving many combinatorial optimization problems including flow shop sequencing problems [124, 116, 97]. Reeves improved computational efficiency of this heuristic and compared it with simulated annealing and found that tabu search method outperforms simulated annealing on a wide variety of problem instances. A list of problems for which Tabu search method has been applied can be found in [51]. For a detailed description of algorithm and the parameter selection refer to chapter four.

Genetic Algorithms

From operations research perspective, genetic algorithms can be viewed as an intelligent exploitation of random search [95]. Though biased sampling heuristics [100, 83], from this perspective, can be seen as precursors to genetic algorithms, GA approach is far more flexible and provides a general framework for a wide variety of problems. Genetic algorithms were developed initially by Holland and his associates (see [95]). The working of a simple genetic algorithm can be explained in the following way. A genetic algorithm works by maintaining a population of chromosomes — potential parents — whose fitness values have been calculated. Each chromosome encodes a solution to the problem and its fitness value is related to the value of objective function corresponding to that solution. Two chromosomes (parents) are selected at random or with a probability proportional to their fitness values or by some other criteria (selection rule). They are then mated (crossed over) by choosing a cross over point X at random and the offspring consisting of a pre-X section of one parent and post-X section of the other parent is formed. Some of the chromosomes in population are eliminated randomly or based on their fitness values in order to propagate better offsprings. Algorithm terminates with a population of solutions when a solution with desired accuracy is found or when the increase in average population fitness is less than certain level or when some other well chosen criteria is met. To maintain diversity among the population and to achieve good results a low probability mutation operator which randomly changes the solution is used. An efficient implementation of genetic algorithm demands careful choice of selection, cross over, and mutation operators. Different types of these operators, successfully used in practice for efficiently solving different problems, are given in the book by Goldberg [53]. A systematic treatment on GAs is contained in the books by Holland [62] and Goldberg [53]. The book by Goldberg also includes a survey of interesting applications carried out in recent times. Davis and Principe [34] give a simulated annealing like convergence theory for simple genetic algorithm.

In flow shop sequencing context, GAs have been mainly used as either heuristics or as a tool (classifier systems) for finding new heuristics. When GA is used as a heuristic, a chromosome usually represents a complete solution i.e, permutation of job orders; and the cross over operators are normally borrowed from traveling salesman problem implementations. When GA is used as classifier systems, a chromosome represents a heuristic and the inherent unsupervised learning aspect of GA is utilized in finding new heuristics. There have been successful attempts at both types of implementations of GAs. Gary [43] used genetic algorithms for scheduling flow shop releases. Reeves [95] reports that GAs implemented for flow shop scheduling give solutions that are comparable with the solutions offered by simulated annealing and

bu search methods. GA based classifier systems were developed by Vinay and
iarles [8], and Hilliard and Liepins [60] for discovering new heuristics.

ther General Methods

The other general methods that are successfully applied to scheduling are knowl-
lge based systems, neural networks and filtered beam search. Most of the scheduling
search in knowledge based systems is concerned with job shop and FMS; the reason
eing that the complexities that arise due to alternative routing, machine flexibili-
es etc., and the dynamic nature of these shops frequently preclude effective static
:heduling [21]. In contrast, dynamic nature of flow shops, which occurs due to
on simultaneous arrival of jobs [99] or due to alternate routing in multistage flow
iops can be incorporated in the formulation of the mathematical model; For ex-
mple flow shops with release times and Parallel flow shops [110]. This convenience
as resulted in the paucity of published literature regarding expert systems for flow
hops. The expert systems that are implemented for scheduling systems differ in the
nowledge acquisition, the knowledge representation and the control mechanism to
rocess knowledge [4].

The structure of an expert system generally contains three components : knowl-
edge database, inference engine and control mechanism [126]. Collection of rules
which represent the experts' knowledge is called the knowledge database. Popular
knowledge representation techniques are logic, rule, frames, networks or combination
of these. Inference engine is a program which uses the knowledge database to pro-
duce an expert recommendation with complete or incomplete information. The most
common structure of an inference engine is the " if then " statement. Control is
the reasoning deduction performance of an inference engine (forward chaining and
backward chaining).

There are many experimental expert systems built for scheduling job shops and
FMS [20, 28, 94, 26, 127, 126]. An expert system with machine learning capabilities

was built by Shaw et al [108]. Ress and Currie [98] discuss the prototyping cycle methodology for expert system development for job shop scheduling and point out the benefits of the methodology. For a recent review of AI based scheduling systems interested readers can refer to [21]. Glover [51] surveys the emerging approaches in heuristic search for solutions to combinatorially complex problems and reports that the most fruitful area in heuristic search is the area combining AI and Operations Research.

There are some Artificial Neural Network (ANN) implementations for solving scheduling problems [103, 109, 117]. Dagli and Sittisathanchai [30] combine genetic algorithms and artificial neural networks for solving job shop scheduling problems. In the above work, ANN was used as an evaluator for finding the fitness of genes generated by genetic algorithm. A beam search which was developed in AI and used by Lowrre and Rubin (refer to the discussion in [87]) for speach and image recognition systems was used for solving scheduling problems by Fox and Ow (seen in [87]). Ow [87] proposes a filtered beam search based on the principle of beam search and compares it with other heuristic methods for flow shop scheduling [87] and finds that filtered beam search method outperforms the heuristic by Campbell et al [19] for solution quality.

1.4 Motivation for and Organization of the Thesis

1.4.1 Motivation

From the above literature survey, it is evident that most of the published research in flow shop scheduling concerns static flow shops with makespan as optimization criterion. It is difficult to surmise reasons for this bias in research. But, one can onjecture at least for modern heuristic techniques for solving combinatorial opti-ıization problems that this problem is treated as a vehicle for demonstrating many a haracteristics of combinatorial optimization problems. Hence the concentrated effort n this problem for makespan criterion. Nevertheless, importance of other criteria or static flow shop scheduling problem or of flow shops with release times cannot be gnored, in operations management context, for efficient running of mass production ystems, which, normally are laid out as flow shops. Hence, static flow shops with otal flow time criterion and flow shops with release time are the main focus of this hesis.

As mentioned earlier, Polynomially solvable special cases of flow shop scheduling problems are of interest not only because they occur in actual flow shops but also because they provide valuable insight for designing heuristics to solve more general problems. For example, the multi stage heuristic based on Johnson's rule given by Campbell et al [19] gives very good results. The SPT heuristic which gives optimal results for single machine mean flow time criterion gives very good results when applied as a dispatching rule to job shop problems. Sometimes we may be able to use the knowledge about special cases for increasing the time efficiency of exponential algorithms, like branch and bound, for solving general problems see chapter 3. Hence it is advantageous to know as many polynomially solvable special cases as possible for a type of problem. In chapter 2, we develop special cases for static flow shop problem with mean flow time criterion and for flow shop problems with release times for makespan criterion. In chapter 3, we study the effect of special cases on a general scheduling method, the branch and bound method.

In the previous sections we have seen that heuristics can be classified into two categories : Problem Specific and General Methods. It is relatively easy to get worst case performance bounds for problem specific heuristics than for general methods. A heuristic with known worst case performance bound tends to instill confidence in the lecision maker as he/she knows what to expect in the worst case and can be prepared for the contingency. But, a heuristic with known worst case bounds or with superior worst case bounds is not guaranteed to exhibit superior average case behavior. So, it is advantageous to have theoretical bounds on both average and worst case performances for deciding upon a heuristic to use for a specific problem. Theoretical results on average case performance are rarely computed owing to the difficulties in defining and analyzing the probability distributions that adequately represent the set of problems to be encountered [91]. Hence, the average case behavior of heuristics are empirically computed using the results obtained on randomly generated problems. There is no published attempt at comparing the general and problem specific heuristic methods for finding out the relative merits of these two. In chapter 4, some problem specific heuristics, an improvement method based on branch and bound, and a hybrid algorithm based on simulated annealing and tabu search, are proposed. These solution techniques are compared with general heuristic methods and other problem specific heuristics. Since static flow shop problem with makespan as optimization criterion is well researched, we have considered flow shop problem with release times, which is more general, (see chapter 4) for this study.

1.4.2 Organization of the Thesis

The Second chapter deals with special cases in flow shop scheduling. In the third chapter, effect of special cases on general scheduling algorithm, branch and bound, is studied and approximate algorithms are considered in chapter four. At the expense of being repetitive, pertinent literature is mentioned at the beginning of each chapter to maintain relative independence of the chapters. Relevant conclusions are also included at the end of each chapter. Chapter five concludes the thesis and discusses the scope for further research.

hapter 2

XACT ALGORITHMS

.1 Introduction

his chapter develops some polynomial algorithms to solve special cases of flow shop oblem defined in the previous chapter. We would like to re-state it in a way which relevant to the material presented in this chapter. According to Garey [44], "A articular instance of an m-machine flow shop problem consists of a collection $\mathbf{T}$ of ains (or jobs), where each chain $t \in \mathbf{T}$ is a sequence of tasks $t[1], t[2], \ldots, t[m]$ and ch task $t[i]$ has a length $\tau(t[i])$ which may be assumed to be non negative integer". hey also define the schedule as a function $S : \{t[i] \mid t \in T, 1 \leq i \leq m\} \longrightarrow Z^+$ hich gives the starting time of each task on an appropriate machine. A nonpremptive chedule, which is of our interest, should, in addition, satisfy the following constraints:

. Only one task is processed on one machine at any given time and once a task egins processing on a machine it is processed till its completion.

. Task $t[i+1]$ should not start before completion of the task $t[i]$ $\forall t \in T$, and $, 1 \leq i \leq m$,

When the sequence of tasks processed on each machine is same, the resulting schedule is known as a permutation schedule [99]. Johnson [67] has proved that when number of machines in a flow shop is restricted to two, permutation schedules dominate nonpermutation schedules for any regular measure of performance. Considerable amount of research effort is spent in designing methods to find permutation schedules which optimize certain chosen criterion. Some of the popular criteria are (a) makespan, (b) mean flow time, (c) maximum tardiness, (d) mean tardiness, and (e) number of tardy jobs.

Most of the problems in this area belong to a group of intractable problems called NP hard problems (see [27, 99]). For these problems, it is highly unlikely that efficient exact algorithms can be designed. In the absence of efficient algorithms for these problems, much of the research is focused on finding good approximate algorithms (see [6, 19, 32, 81, 88, 89]) or on designing controlled enumeration methods like branch and bound (see [64, 75, 74, 77]). Some of the researchers have developed efficient algorithms to solve some of the special cases of the above problems. Notable among them are [2, 5, 67, 106, 114]. Most of these special cases have makespan as their optimization criterion. But for the restrictive special cases mentioned by Ignal and Schrage [64] there has been pronounced absence of efficient algorithms for solving problems with mean flow time as optimization criterion. The efforts of the section 2.2 are directed to fill this gap. Since two machine flow shop problem with mean flow time as optimization criterion is proven to be NP hard, only special cases of two machine flow shop are considered in this work.

Much of the work done in flow shop scheduling is with the assumption that all the jobs are available at the beginning of the scheduling period i.e., the job release times are zero. The two machine flow shop problem with non zero release times is proven to be NP hard [24]. Garbowski [45] presented a branch and bound algorithm for solving this problem. Potts [92] presented five heuristics and their worst case performance analysis. But for these scattered attempts, this problem has received very little attention from researchers. In the second section of this chapter we present the special cases of this problem.

The section 2.3 considers the two machine flow shop problem with release times. Consider a flow shop problem with two machines and n jobs. All jobs are *not* ready at time $t = 0$ and each job arrives at a definite time. Each machine can process only one job at a time and no preemption is allowed. All jobs have to go through the two machines in the same order. The problem is how to arrange the jobs such that the schedule length is minimized.

An interesting application of the above dynamic flow shop problem is the special case of flexible flow shop problem given in [110]. Consider this flexible flow shop $F3/n, 1, 1/C_{max}$. Since all the jobs get processed simultaneously in machining center one, the sequencing of the jobs is limited to machining centers two and three with completion time of a job in machining center one as the job's ready time. Since machining centers two and three have only one machine each, this problem becomes a deterministic dynamic flow shop problem with two machines.

2.1.1 Notations

General notation followed throughout this chapter is as follows:

$t_i[0]$: Release time of i^{th} job.
$t_i[1]$: Processing time of i^{th} job on the first machine.
$t_i[2]$: Processing time of i^{th} job on the second machine.
C_i: Completion time of i^{th} job.
Let
Machine k dominates machine l imply $\min(t_i[k]) \geq \max(t_i[l])$.
J minimal [23] flow shop imply that each job i has its least processing time on machine J.

2.2 Special cases of $n|2|F|\bar{F}$ Problem

The cases we consider in this work are as follows:

(a) When machine two dominates machine one,

(b) When machine one dominates machine two,

(c) One minimal flow shop i.e. every job has its least processing time on the first machine, and

(d) Two minimal flow shop [23] i.e. every job has its least processing time on the second machine.

The first two cases are similar in construction to the cases constructed by Mukhopadhyay and Johnson [5, 67] for three machine flow shop problem with makespan as optimization criterion. The last two cases are similar in construction to the cases constructed by Burns [15] and Achugbe [2] for three machine flow shop problem with makespan as optimization criterion.

2.2.1 Notations

The notations that are specific to this section are:
X_i: Idle time of machine two before processing i^{th} job and after processing $(i-1)^{th}$ job.
Y_i: Idle time of i^{th} job in buffer between first and second machines.

$$X_1 = t_1[1]$$

$$C_1 = X_1 + t_1[2]$$

$$X_2 = \max(-(X_1 + t_1[2]) + (t_1[1] + t_2[1]), 0)$$

$$C_2 = X_1 + X_2 + t_1[2] + t_2[2]$$

$$X_3 = \max(-\sum_1^2 X_i - \sum_1^2 t_i[2] + \sum_1^3 t_i[1], 0)$$

$$C_3 = \sum_1^3 X_i + \sum_1^3 t_i[2]$$

$$\sum_{i=1}^{n} \mathrm{C}_i = \sum_{j=1}^{n} \left[\sum_{i=1}^{j} \mathrm{X}_i + \sum_{i=1}^{j} \mathrm{t}_i[2] \right]$$

$$\sum_{i=1}^{n} \mathrm{C}_i = \sum_{j=1}^{n} (n - j + 1)(X_j + t_j[2]) \tag{2.1}$$

2.2.2 Special Case: machine two dominates machine one, $\min(t_i[2]) \geq \max(t_i[1])$

The cases in this section and next section are similar to the cases constructed by Johnson and Mukhopadhyay [67, 5].

Claim 1 *For the above mentioned special case, for a given first job, SPT ordering on second machine gives optimal sequence for the remaining (n-1) jobs.*

Proof:

$$X_1 = t_1[1]$$

$$X_{k|2\leq k\leq n} = \max\left(\sum_{i=1}^{k} t_i[1] - \sum_{i=1}^{k-1} X_i - \sum_{i=1}^{k-1} t_i[2], 0\right)$$

$$X_{k|2\leq k\leq n} = \max\left(\sum_{i=2}^{k} t_i[1] - \sum_{i=2}^{k-1} X_i - \sum_{i=1}^{k-1} t_i[2], 0\right)$$

Since $\min(t_i[2]) \geq \max(t_i[1])$,

$X_k = 0$ for $2 \leq k \leq$ n.

Therefore the equation 2.1 can be written as

$$\sum_{i=1}^{n} C_i = nX_1 + \sum_{j=1}^{n}(n-j+1)t_j[2] \tag{2.2}$$

$$\text{Total completion time } = (nt_1[1] + nt_1[2]) + \left(\sum_{j=2}^{n}(n-j+1)t_j[2]\right)$$

The second term can be minimized by arranging the jobs 2 to n in non decreasing order of their processing times on second machine. Hence the claim.

Therefore a sequence minimizing $\bar{F}$ can be obtained by the following procedure:

Step 1. Select each job as the first job and arrange the remaining jobs in SPT order on the second machine. This step gives us n distinct sequences.

Step 2. The best out of the n sequences generated in the previous step is the optimal sequence.

Complexity Analysis:

1. Initial ordering of the jobs on their second machine takes $O(n \log n)$ time
2. Rearranging of every sequence takes $O(n)$ time. There are n sequences. Therefore, it takes $O(n^2)$ time.
3. Finding out the best sequence takes $O(n)$ time.

Total time complexity = $O(n \log n) + O(n^2) + O(n) = O(n^2)$.
Therefore the algorithm runs in $O(n^2)$ time.

i	$t_i[1]$	$t_i[2]$
1	20	23
2	18	20
3	16	29
4	8	35
5	17	21

Table 2.1: A two machine five job flow shop problem with $\min(t_i[2]) \geq \max(t_i[1])$

First job	SPT sequence	Total flow time
1	2,5,3,4	451
2	5,1,3,4	436
3	2,5,1,4	449
4	2,5,1,3	433
5	2,1,3,4	432

Table 2.2: Solutions generated by the algorithm to the problem given in the previous table

2.2.2.1 Example:

Consider the five job problem as given in table 2.1.

Solutions generated by the algorithm are given in table 2.2.

The optimal sequence is 5,2,1,3,4 with a total flow time of 432 units.

2.2.3 Special Case: machine one dominates machine two, $\min(t_i[1]) \geq \max(t_i[2])$

Claim 2 *For the above mentioned special case SPT ordering on first machine gives optimal sequence.*

Proof: since $\min(t_i[1]) \geq \max(t_i[2])$, no job needs to wait in the buffer between first machine and second machine. Therefore completion time of a job i in a given

i	$t_i[1]$	$t_i[2]$
1	23	20
2	20	18
3	29	16
4	35	8
5	21	17

Table 2.3: A two machine five job flow shop problem with $\min(t_i[1]) \geq \max(t_i[2])$

permutation is sum of the processing times on first machine of the jobs preceding job i plus sum of the processing times of job i on both the machines. Therefore, it is easy to see that for any permutation $[1], [2], \ldots, [n]$ the total flow time is given by

$$\sum_{i=1}^{n}(n-i+1)t_i[1] + \sum_{i=1}^{n} t_i[2] \tag{2.3}$$

The second term in the above expression is constant and first term can be minimized by a permutation which is in SPT ordering on first machine. Hence the claim.

2.2.3.1 Example:

Consider the problem given in table 2.3

Solution: The SPT sequence on the first machine is 2,5,1,3,4 with a total flow time of 425 units.

2.2.4 Special Case: two minimal flow shop, $t_i[1] \geq t_i[2]$ $\forall$ $1 \leq i \leq n$

Theorem 1 *For the above case, the permutation schedule given by SPT rule on first machine minimizes mean flow time.*

Proof: This theorem is proved as a consequence of following claim.

Claim 3 *If we follow SPT ordering on machine one, then job waiting time in the buffer between the two machines, Y_i, is zero for every job.*

Proof: By induction.
For the first job, the job waiting time in the buffer, $Y_1 = 0$
Suppose job waiting time is 0 for all jobs up to some k, $1 \leq k <$ n.
Then machine two becomes available for $(k+1)^{th}$ job at time
$\sum_{i=1}^{k} t_i[1] + t_k[2]$
and job $(k+1)^{\text{th}}$ finishes its processing on machine one at $\sum_{i=1}^{k+1} t_i[1]$.
Since $t_{k+1}[1] \geq t_k[1] \geq t_k[2]$,
$\sum_{i=1}^{k+1} t_i[1] = \sum_{i=1}^{k} t_i[1] + t_{k+1}[1] \geq \sum_{i=1}^{k} t_i[1] + t_k[2]$, the time at which second machine is available for $(k+1)^{\text{th}}$ job.
Therefore $(k+1)^{\text{th}}$ job does not wait in the buffer between the two machines. Hence the claim.
From above claim, for a schedule based on SPT ordering on first machine, the total flow time is given by
$\sum_{i=1}^{n}(n-i+1)t_i[1] + \sum_{i=1}^{n} t_k[2]$.
Since the first term is minimized by SPT and second term is constant, the total flow time for any schedule given by some rule other than SPT will be greater than or equal to that given by SPT. Q.E.D.
The optimal sequence for the above special case can be obtained in $O(n \log n)$ time.

2.2.4.1 Example:

Consider the problem given in the table 2.4.

Solution: The SPT sequence on the first machine is 2,3,5,1,4 with a total flow time of 470 units.

i	$t_i[1]$	$t_i[2]$
1	35	23
2	18	16
3	22	22
4	43	11
5	27	26

Table 2.4: A two machine five job flow shop problem with $t_i[1] \geq t_i[2] \forall i$, $1 \leq i \leq n$

2.2.5 Special Case: one minimal flow shop, $t_i[2] \geq t_i[1]$ $\forall$ i, $1 \leq i \leq n$

We prove that for the above case the problem is NP hard. We prove this result by reducing the 3-partition problem to our candidate decision problem. We state the optimization problem as decision problem and also present 3-partition problem. For ease of understanding, the notation of Garey [44] is followed in this section

Special case of 2-machine Flow shop problem: Given a collection **C** of chains for 2-machine flow shop, with task lengths given by $\tau : \{c[i] | c \in \mathbf{C}, 1 \leq i \leq 2, c[2] > c[1]\} \longrightarrow Z^+$ and given a bound D, does there exist a schedule S for **C** whose total flow time does not exceed D ?
3-partition problem [10]: Given positive integers n, B and a set of integers $A = \{a_1, a_2, \ldots.., a_{3n}\}$ with $\sum_{i=1}^{3n} a_i = nB$ and $B/4 < a_i < B/2$ for $1 \leq i \leq 3n$, Does there exist a partition $\langle A_1, A_2, \ldots., A_n \rangle$ of A into 3-element sets such that, for each $i, \sum_{a \in A_i} a = B$?

Theorem 2 *The above mentioned special case of two machine flow shop problem is NP complete.*

Proof: Proof of this theorem is given as a consequence of series of claims. Suppose we are given a 3-partition problem n, B and $A = \{a_1, .., a_{3n}\}$. First the details of corresponding flow shop are described:

Let

$$u = 3nB + 1$$

$$v = u + 3nB + \mathrm{nu}/3 - 2\mathrm{nuB}/3 + [n^2u/2 + n^3u/6](B + 1) + v[(n^3u^2 - 4\mathrm{nu}^2)/6]$$

$$t = uv + B + 1$$

$$x = (n(n + 1) + 2n + 2)t + v$$

Consider the following three types of chains of tasks for scheduling so as to minimize the mean flow time:

T type chains:

$$\tau(T_i) = \{\mathrm{it}, \mathrm{it} + 1\} \forall 0 \leq i \leq n$$

X type chains:

$$\tau(X_i) = \{0, x_i\} \forall 1 \leq i \leq v$$

U type chains which are further divided into two sub types V and W.

$$\tau(V_{i,j}) = \{0, v\} \text{ for } 1 \leq i \leq n, 1 \leq j \leq u - 3$$

$$\tau(w_i) = \{0, v + a_i\} \text{ for } 1 \leq i \leq 3n$$

The bound on mean flow time is given by $D_{\in} = T_{\in} + X_{\in} + U_{\in}$, where

$$\begin{aligned} T_{\in} &= \left[\sum_{i=0}^{n}\left(\sum_{j=1}^{i} jt\right) + it + 1\right] \\ &= n(n + 1)(2n + 1)t/12 + n(n + 1)t/4 + n(n + 1)t/2 + n \end{aligned} \tag{2.4}$$

$$\begin{aligned}
X_{\in} &= \sum_{i=1}^{v} (n(n+1)t/2 \quad + \quad nt \quad + \quad 1 \quad + \quad \text{ix}) \\
&= (n(n+1)t + 2nt + 2)v/2 + v(v+1)x/2 \\
U_{\in} &= 3nB + \sum_{i=0}^{n-1} \left[\sum_{j=1}^{u} (i(i+1)t/2 \quad + \quad \text{it} \quad + \quad 1 \quad + \quad jv) \right] \\
&= \text{nu}(\text{nu}+1)v/2 + 3nB + \text{nu}/3 - 2\text{nuB}/3 + n^2u(B+1)/2 + n^3u(B+1)/6 \\
&\quad + v(n^3u^2 - 4\text{nu}^2)/6
\end{aligned}$$

Let

$k = 3nB + \text{nu}/3 - 2\text{nuB}/3 + n^2u(B+1)/2 + n^3u(B+1)/6$ and

$f(v) = v(n^3u^2 - 4\text{nu}^2)/6$

then

$U_{\in} = \text{nu}(\text{nu}+1)v/2 + k + f(v)$

$v = u + k + f(v).$

It can be easily seen that the task lengths and the time taken to construct the input are bounded by a polynomial in nB.

Now we show that the desired partition exists if and only if there is a feasible schedule for **C** with the $\bar{F} \leq D_{\in}$.

Suppose that there is a partition $< A_1, A_2, \ldots, A_n >$ of desired form. That is, each set $A_i = \{a_{g(i,1)}, a_{g(i,2)}, a_{g(i,3)}\}$ such that $\sum_{j=1}^{3} a_{g(i,j)} = B$ for $1 \leq i \leq$ n.

Then the correspondin; schedule is given by:

$S(T_i) = \langle i(i-1)t/2, i(i+1)t/2 \rangle$ for $0 \le i \le$ n

$S(X_i) = \langle 0, n(n+1)t/2 + nt + 1 +$ ix$\rangle$ for $1 \le i \le v$

$S(V_{(i,j)}) = \langle 0, i(i-1)t/2 + (i-1)t + 1 + (j-1)v \rangle$ for $1 \le i \le n$ and $1 \le j \le u-3$

$S(W_{g(i,1)}) = \langle 0, i(i-1)t/2 + (i-1)t + 1 + (u-3)v \rangle$ for $1 \le i \le n$

$S(W_{g(i,2)}) = \langle 0, i(i-1)t/2 + (i-1)t + 1 + (u-2)v + a_{g(i,1)} \rangle$ for $1 \le i \le n$

$S(W_{g(i,3)}) = \langle 0, i(i-1)t/2 + (i-1)t + 1 + (u-1)v + a_{g(i,1)} + a_{g(i,2)} \rangle$ for $1 \le i \le n$

It can be easily seen that T type task chains and X type task chains are scheduled in SPT order and u of U type task chains are scheduled between finish time of $(i-1)^{th}$ T type task and start of i^{th} T type task. Sum of the task lengths of these u U type tasks is $(t-1)$ which equals the time between finish time of $(i-1)^{th}$ T type task and start of i^{th} T type task. It can be easily verified that the above schedule is a valid schedule. Now we prove that the $\bar{F}$ is less than $D_{\in}$.

The sums of finish times of T type tasks and X type tasks are $T_{\in}$ and $X_{\in}$ respectively, whereas sum of finish times of U type tasks, U_f, is less than $U_{\in}$.

$$\begin{aligned} U_f &= \sum_{i=0}^{n-1} \left[\begin{array}{l} \sum_{j=1}^{u} (i(i+1)t/2 \; + \; \text{it} \; + \; 1 \; + \; jv) \\ +3a_{g(i+1,1)} + 2a_{g(i+1,2)} + a_{g(i+1,3)} \end{array} \right] \\ &< \sum_{i=0}^{n-1} \left[\sum_{j=1}^{u} (i(i+1)t/2 \; + \; \text{it} \; + \; 1 \; + \; jv) + 3 \sum_{i=1}^{3n} a_i \right] = D_{\in}. \end{aligned} \tag{2.5}$$

Therefore the above constructed schedule is a good (feasible) schedule.

Now we prove that if there exists a good schedule then there must be a desired partition of A. This is proved using the following series of claims on chains of tasks and on schedule length.

Claim T_1: Every task which has task length zero on P_1 starts at time zero on P_1.

Claim T_2: Since $T_i[1] < T_{i+1}[1]$ and $T_i[2] < T_{i+1}[2]$, $S(T_i[1]) \leq S(T_{i+1}[1])$ for $0 \leq i \leq n$, $T_{i+1}[1]$ starts as soon as $T_i[1]$ finishes.
Therefore $S(T_i[1]) = i(i+1)t/2$

Claim T_3: The order of executing the tasks on P_1 and P_2 can be the same. It was proved by Johnson [67] that for any two machine problem, the order on the two machines can be same without any increase in the value of measure of performance, for any regular measure of performance. Thus

$$S(T_i[2]) \leq S(T_{i+1}[2]) \forall 0 \leq i \leq n-1$$

Claim X_1: Since all X type chains have same length, $S(X_i[2]) \leq S(X_{i+1}[2]) \ \forall \ 1 \leq i \leq v-1$.

Claim X_2: There exists a good schedule satisfying previous claims and such that no task $X_i[2]$ starts before $n(n+1)t/2 + nt + 1$

Proof: Let S be the minimum mean flow time schedule satisfying the previous claims but violating claim X_2. Then let X_i be the task which starts before $n(n+1)t/2+nt+1$. From claim X_1 and the length of X type chains, $X_1[2]$ is the only task that can be executed before $n(n+1)t/2 + nt + 1$.

Since finish time of $X_1[2]$ will be at least $(n(n+1) + 2n + 2)t + v$, all other tasks can finish their processing on machine P_1 before completion of $X_1[2]$. Tasks executed on machine P_2 after $X_1[2]$ can be reordered without violating chain constraint. In order

.inimize mean flow time they must be executed shortest first in S. We will bound time and show that it cannot be a good schedule.

$$\begin{aligned} X_l &= x + \sum_{i=1}^{v-1}(n(n+1)t/2 + nt + x + ix) & (2.6) \\ &= X_\in - n(n+1)t/2 - nt - 1 \end{aligned}$$

ce $T_n[2]$ cannot be executed till task $X_1[2]$ is finished, the lower bound on the T e chains is given by

$$\begin{aligned} T_l &= \left[\sum_{i=0}^{n-1}\left(\sum_{j=1}^{i} jt\right) + \text{ it } + 1\right] + x + nt + 1 & (2.7) \\ &= T_\in + n^2t/2 + 3nt/2 + 2t + v + 1 \end{aligned}$$

mple lower bound for completion time of U type tasks is

$$U_l = \sum_{i=1}^{\text{nu}} \text{iv} = \text{nu}(\text{nu}+1)v/2 = U_\in - k - f(v)$$

he lower bound on the schedule length D_l is

$$\begin{aligned} D_l &= U_l + T_l + X_l & (2.8) \\ &= U_\in - k - f(v) + T_\in + n^2t/2 + 3nt/2 + 2t + v + 1 \\ &\quad + X_\in - n(n+1)t/2 - nt - 1 \end{aligned}$$

ince v is greater than $k + f(v)$,

$D_l > D_\in$.

Therefore, S is not a good schedule.

Claim X_3: There exists a good schedule satisfying the previous claims such that $S(X_1[2]) = n(n+1)t/2 + nt + 1$. In addition P_2 is not idle before $n(n+1)t/2 + nt + 1$ and T and U type tasks are finished before $n(n+1)t/2 + nt + 1$.

Proof: Let S be a good schedule satisfying previous claims but not X_3.

Suppose $S(X_1[2]) > n(n+1)t/2 + nt + 1$.

Then the bounds are

$$T_l = T_\in,$$

$$\begin{aligned} X_l &= \sum_{i=1}^{v}(n(n+1)t/2 + nt + 2 + \mathrm{ix}) \\ &= X_e + v, \ \ and \end{aligned} \tag{2.9}$$

$U_l = \mathrm{nu(nu{+}1)}v/2 = U_\in - k - f(v)$

The lower bound on schedule length is

$$\begin{aligned} D_l &= T_l + X_l + U_l \\ &= T_\in + X_\in + U_\in + v - k - f(v) > D_\in \end{aligned} \tag{2.10}$$

herefore, S is not a good schedule.

ince sum of the lengths of T, V and W type tasks is $n(n+1)t/2 + nt + 1$; for P_2 e to idle, one of these tasks should be executed after $n(n+1)t/2 + nt + 1$. Since elayed task is smaller than $X_1[2]$, we can get a valid schedule with lesser mean flow me by interchanging the two, a contradiction.

Claim U_1: There exist a good schedule satisfying the previous claims such that $t_i[2]$ is preceded by exactly iu U type tasks for $0 \leq i \leq n-1$.

Proof: Let S be a good schedule satisfying the previous claims but not U_1.

Case a: The total number of tasks is less than iu. Then the total length of the tasks preceding i will be no more than

$$i(i-1)t/2 + (iu-1)v + nB < i(i+1)t/2$$

By chain constraints and claim T_2 $S(t_i[2]) \geq i(i+1)t/2$. Therefore, P2 must be idle before $i(i+1)t/2$, hence before $n(n+1)t/2$ violation of claim X_3.

Case b: Total number of tasks greater than iu. The total length of the tasks preceding i must be at least

$i(i-1)t/2 + (iu+1)v = i(i+1)t/2 + v - iB$ Therefore, $S(t_i[2]) \geq \imath(i+1)t/2 + v - iB$ Lower bound on T type tasks is

$$\begin{aligned} T_l &= \sum_{j=1}^{n}(j(j+1)t/2 + jt + 1 + v - iB) && (2.11)\\ &= T_\in + v - iB \end{aligned}$$

Since sum of the X type tasks is at least $X_\in$ by the preceding claims and sum of U type tasks is at least $U_\in - K - f(v)$,

$$\begin{aligned} D_l &= T_l + X_\in + U_l && (2.12)\\ &= T_\in + v - iB + X_\in + U_\in - k - f(v) \end{aligned}$$

Since $v = u + k + f(v)$ and $U > iB$, $D_l > D_\in$

Therefore, S is not a good schedule.

Claim U_2: There exists a good schedule satisfying the previous claims such that

$$S(T_i[2]) = i(i+1)t/2 \forall 0 \leq i \leq n$$

Proof: Let S be a good schedule satisfying previous claims but not U_2.
By chain constraint T_2,
$S(T_i[2]) \geq i(i+1)t/2$. Suppose for any i,

$S(T_i[2]) > i(i+1)t/2$,
then by claim X_3, i must be such that $0 \leq i \leq n-1$. By claim U_1 there are u U type tasks between $T_i[2]$ and $T_{i+1}[2]$. Sum of the finish times of these tasks must be
$\sum_{j=1}^{u} (i(i+1)t/2 \quad +\text{it} \quad +2 \quad +jv)$,
whereas sum of the finish times of the U type tasks between $T_k[2]$ and $T_{k+1}[2]$ for any $k, k \neq i$ and $0 \leq k \leq n$ must be at least
$\sum_{j=1}^{u} (k(k+1)t/2 \quad +kt \quad +1 \quad +jv)$.

Sum of the finish times of U type tasks is

$$\sum_{k=0}^{n-1} \left(\sum_{j=1}^{u} (k(k+1)t/2 \quad +kt \quad +1 \quad +jv) \right) + u = U_{\in} + 1$$

Since sums of the finish times of X type and T type chains are at least $X_{\in}$ and $T_{\in}$, respectively.

$$\begin{aligned} D_l &= T_{\in} + X_{\in} + U_l && (2.13) \\ &= T_{\in} + X_{\in} + U_{\in} + 1 \\ &> D_{\in}, \text{a contradiction.} \end{aligned}$$

Proof of theorem 2: Let S be a good schedule satisfying previous claims. Then the desired partition is given by the following sets A_i, $1 \leq i \leq n$:

$$A_i = \{a_k | S(T_{i-1}[2]) < S(w_k[2]) < S(T_i[2])\} 1 \leq i \leq n$$

Now we show that A_i contains exactly three elements and their sum is equal to B.

By preceding claims,
$U_i = \{u \in U | S(T_{i-1}[2]) < S(u[2]) < S(T_i[2])\}$
has cardinality u and
$\sum_{u \in U} \tau(u[2]) = t - 1 = uv$+B.
Every V type task has length v and every W type task has length between $v + B/4$ and $v + B/2$. This means that every U_i must contain exactly three w type tasks and that
$\sum_{w_k \in U_i} \tau(w_k[2]) = 3v$+B.
Hence $\sum_{a_k \in U_i} a_k$ =B.
Thus all A_i are disjoint three element subsets each of which has sum exactly B and therefore, $\langle A_1, A_2, \ldots, A_n \rangle$ is our desired partition.

2.3 Special Cases of $n|2|F, r_i \geq 0|C_{max}$ Problem

2.3.1 Notations

The notation used in this section is at slight variation from that in the previous section because there can be idle time on both first and second machines.
Let

X_i be the idle time on machine 1 before the processing of job i, and
Y_i be the idle time on machine 2 before the processing of job i.

Then, we have,

$$X_1 = t_1[0].$$

$$Y_1 = X_1 + t_1[1].$$

$$X_2 = \max(t_2[0] - (X_1 + t_1[1]), 0).$$

$$Y_2 = \max(t_1[1] + t_2[1] + X_1 + X_2 - t_1[2] - Y_1, 0).$$

$$X_1 + X_2 = \max(t_2[0] - t_1[1], X_1).$$

$$Y_1 + Y_2 = \max(t_1[1] + t_2[1] + X_1 + X_2 - t_1[2], Y_1).$$

$$X_3 = \max(t_3[0] - X_2 - t_2[1] - X_1 - t_1[1], 0).$$

$$Y_3 = \max(t_1[1] + t_2[1] + t_3[1] + X_1 + X_2 + X_3 - t_1[2] - Y_1 - t_2[2] - Y_2, 0).$$

$$X_1 + X_2 + X_3 = \max(t_3[0] - \sum_{i=1}^{2} t_i[1], \sum_{i=1}^{2} X_i).$$

$$Y_1 + Y_2 + Y_3 = \max(\sum_{i=1}^{3} t_i[1] - \sum_{i=1}^{2} t_i[2] + \sum_{i=1}^{3} X_i, \sum_{i=1}^{2} Y_i).$$

$$\sum_{i=1}^{n} X_i = \max(t_n[0] - \sum_{i=1}^{n-1} t_i[1], \sum_{i=1}^{n-1} X_i).$$

$$\sum_{i=1}^{n} Y_i = \max(\sum_{i=1}^{n} t_i[1] - \sum_{i=1}^{n-1} t_i[2] + \sum_{1}^{n} X_i, \sum_{i=1}^{n-1} Y_i).$$

$$\sum_{i=1}^{n} Y_i = \max_{1 \leq v \leq n} \left(H_v + \max_{1 \leq u \leq v} K_u \right)$$

Where

$$H_v = \sum_{i=1}^{v} t_i[1] - \sum_{i=1}^{v-1} t_i[2], \text{ and}$$

$$K_u = (t_u[0] - \sum_{i=1}^{u-1} t_i[1]) \tag{2.14}$$

2.3.2 Special Case: machine one dominates release times, $\min(t_i[1]) \geq \max(t_i[0])$

Let S be the schedule with some arrangement of jobs and let S' be the schedule with jobs j and j + 1 interchanged from schedule S. The schedule S will be better than S' if $\sum_{i=1}^{n} Y_i < \sum_{i=1}^{n-1} Y_i$.

When the schedule is changed from S to S', the $\sum_{i=1}^{n} Y_i$ and $\sum_{i=1}^{n} Y_i'$ differ only in the terms that contain j and $j+1$ as coefficients. Therefore schedule S will be superior to S' if

$$\begin{aligned}\max(H_{j+1} + \max_{1\leq u\leq j+1} K_u, H_j + \max_{1\leq u\leq j} K_u) &< \\ \max(H'_{j+1} + \max_{1\leq u\leq j+1} K'_u, H'_j + \max_{1\leq u\leq j} K'_u)\end{aligned} \tag{2.15}$$

The inequality 2.15 depends not only on j and $j+1$ terms but also on the preceding terms because of the $\max K_u$ and $\max K'_u$ terms.
Since we assume that $\min(t_i[1]) \geq \max(t_i[0])$,
$\max_{1\leq u\leq v} K_u = \max_{1\leq u\leq v}(t_u[0] - \sum_{i=1}^{u-1} t_i[1])$ will always be equal to $t_1[0]$ for $u = 1$ to n.

Under this assumption, the inequality 2.15 becomes

$$\max(H_{j+1} + t_1[0], H_j + t_1[0]) < \max(H'_{j+1} + t_1[0], H'_j + t_1[0])$$

By subtracting $t_1[0]$ from inequality 2.15 and substituting the expression for H_j in it, we get:

$$\begin{aligned}\max(\textstyle\sum_{i=1}^{j+1} t_i[1] - \sum_{i=1}^{j} t_i[2], \sum_{i=1}^{j} t_i[1] - \sum_{i=1}^{j-1} t_i[2]) &< \\ \max(\textstyle\sum_{i=1}^{j+1} t_i[1] - \sum_{i=1}^{j} t_i[2], \sum_{i=1}^{j} t_i[1] - \sum_{i=1}^{j-1} t_i[2])'\end{aligned} \tag{2.16}$$

First Job	Johnson's Sequence	Make Span
1	2,3,5,4	58
2	3,1,5,4	54
3	2,1,5,4	55
4	2,3,1,5	63
5	2,3,1,4	60

Table 2.6: Solutions for the problem given in the previous table

2.3.3 Special Case: machine two dominates machine one, $\min(t_i[2]) \geq \max(t_i[1])$

The equation 2.14 can be written as

$$K_u = \sum_{i=1}^{u} t_i[0]' - \sum_{i=1}^{u-1} t_i[1]$$

where $t_1[0]' = t_1[0] - 0$ and
$t_i[0]' = t_i[0] - t_{i-1}[0]\ \forall i, 2 \leq i \leq n.$

$$H_v = \sum_{i=1}^{v} t_i[1] - \sum_{i=1}^{v-1} t_i[2]$$

Since $\min(t_i[2]) \geq \max(t_i[1])$,

$$H_1 \geq H_2 \geq \cdots \geq H_n$$

Hence the term

$$\max_{1\leq u\leq v\leq n} \{H_v + K_u\} = \max_{1\leq u\leq v\leq n} \{H_v + K_v\}$$

The inequality 2.15 can be written as

$$\max(H_{j+1} + K_{j+1}, H_j + K_j) < \max\left(H'_{j+1} + K'_{j+1}, H'_j + K'_j\right)$$

y subtracting

$$\left(\sum_{i=1}^{j+1} t_i[1] - \sum_{i=1}^{j-1} t_i[2] + \sum_{i=1}^{j+1} t_i[0]' - \sum_{i=1}^{j-1} t_i[1]\right)$$

rom both sides we get,

$$\max(-t_j[2] - t_j[1], -t_{j+1}[1] - t_{j+1}[0]') < \max(-t_{j+1}[2] - t_{j+1}[1], -t_j[1] - t_j[0]')$$

$$\Longrightarrow \min(t_{j+1}[2] + t_{j+1}[1], t_j[1] + t_j[0]') < \min(t_j[2] + t_j[1], t_{j+1}[1] + t_{j+1}[0]')$$

Since, $t_i[0]'$ $\forall i$, $2 \leq i \leq n$ depends on the job that immediately precedes it and can be dynamically calculated as the construction of sequence progresses, we call the above rule as dynamic Johnson's rule.

The following algorithm generates an optimal solution.

Step 0: Let $t_i[0]' = t_i[0], k = 1$, and partial sequence, $\sigma = \{\}$.

Step 1: Calculate $t_i[0]' + t_i[1]$ for all $i \notin \sigma$.

Step 2: Let $\bar{J}$ denote the set of jobs whose $(t_i[0]' + t_i[1]) < (t_i[1] + t_i[2])$ and J denote the set of jobs whose $(t_i[0]' + t_i[1]) \geq (t_i[1] + t_i[2])$. If $\mid \bar{J} \mid \geq 1$, then schedule the job with $min(t_i[0]' + t_i[1])$ from $\bar{J}$ as k^{th} job else schedule the job with $\max(t_i[1] + t_i[2])$ from J as the k^{th} job.
$\sigma = \sigma \cup [k]$ and $k = k + 1$.

Step 3: If $k = n$, goto step 5.

Step 4: Calculate $t_i[0]' = t_i[0] - t_{[k-1]}[0]$ for all $i \notin \sigma$ and goto step 1.

Step 5: Schedule the remaining job as the n^{th} job and stop.

i	$t_i[0]$	$t_i[1]$	$t_i[2]$
1	86	37	53
2	65	40	62
3	56	1	61
4	21	27	63
5	72	23	65

Table 2.7: A two machine five job flow shop problem with $\min(t_i[2]) \geq \max(t_i[1])$

2.3.3.1 Example:

Consider the problem given in table 2.7.

Solution:

Iteration 1: $\bar{J} = [4,3], k = 1, J = [1,2,5], \sigma = \{\}$

Iteration 2: $\bar{J} = [3,5], k = 2, J = [1,2], \sigma = \{4\}$

Iteration 3: $\bar{J} = [2,5], k = 3, J = [1], \sigma = \{4,3\}$

Iteration 4: $\bar{J} = [1,2], k = 4, J = [], \sigma = \{4,3,5\}$

Iteration 5: $\bar{J} = [1], k = 5, J = [], \sigma = \{4,3,5,2\}$

The optimal sequence is {4,3,5,2,1} with a makespan of 352 units.

2.3.4 Special Case: machine one dominates machine two, $\min(t_i[1]) \geq \max(t_i[2])$

Lemma 1 *A job completing its operation on machine one always finds the machine two free.*

Proof: Since $\min(t_i[1]) \geq \max(t_i[2])$, even if two jobs are simultaneously started on machine one and two the time for which machine two is busy in processing the job is less than or equal to that for machine one. Hence any job completing its operation on machine one finds the machine two free.

Consequence: A direct consequence of the above lemma is that the makespan of any permutation schedule can be written as

i	$t_i[0]$	$t_i[1]$	$t_i[2]$
1	86	53	37
2	65	62	40
3	56	61	1
4	21	63	27
5	72	65	23

Table 2.8: A two machine five job flow shop problem with $\min(t_i[1]) \geq \max(t_i[2])$

$$C_{[n]}[1] + t_{[n]}[2] = C_{[n-1]}[1] + \max(t_{[n]}[0], C_{[n-1]}[1]) + t_{[n]}[1] + t_{[n]}[2]$$

Theorem 3 *When the above constraint on the processing times holds, sequencing the first $n-1$ jobs according to Earliest Release Time (ERT) rule will give the minimum makespan for a given n^{th} job.*

Proof: The term $C_{[n-1]}[1]$ is composed of two terms as given below:

$$C_{[n-1]}[1] = \sum_{i=1}^{n-1} X_i + \sum_{i=1}^{n-1} t_i[1]$$

For any sequence of the jobs $\sum_{i=1}^{n-1} t_i[1]$ is constant and $\sum_{i=1}^{n-1} X_i$ is the sum of the idle times caused by the non arrival of jobs at machine one when the machine is free to process the jobs. Hence $\sum_{i=1}^{n-1} X_i$ can be minimized by sequencing the jobs according to non decreasing order of their release times (Earliest Release Time rule). Therefore $\sum_{i=1}^{n-1} X_i$ is minimum for ERT sequence of the $n-1$ jobs.

Consequence: Immediate consequence of the above theorem is that by choosing each job as the n^{th} job and applying ERT rule for the remaining $n-1$ jobs, we get n sequences. At least one among these n sequences is an optimal sequence.

2.3.4.1 Example:

Consider the five job problem shown in table 2.8.

Solution: The solution is shown in table 2.9.

The optimal sequence is 4,2,5,1,3 with a makespan of 326 units.

S.No	ERT Sequence	Last job	Makespan
1	4,3,2,5	1	362
2	4,3,5,1	2	365
3	4,2,5,1	3	326
4	3,2,5,1	4	387
5	4,3,2,1	5	348

Table 2.9: Solutions to the problem given in the previous table

2.3.5 Special Case: two minimal flow shop, $t_i[1] \geq t_i[2]$ $\forall$ i, $1 \leq i \leq n$

Theorem 4 *A two machine flow shop with release times and with makespan as optimization criteria is NP hard in the strong sense even when the processing times are restricted by the constraint $t_i[1] \geq t_i[2] \forall i$, $1 \leq i \leq n$.*

Proof: Let $\langle a_1, a_2, \cdots, a_m \rangle, m = 3n$ and B define an instance of the 3-partition problem such that $\sum_{i=1}^{m} a_i = nB$ and $B/4 < a_i < B/2$. For this instance, we construct the following two machine $(m + 2n + 1)$ job flow shop decision problem instance:

$t_i[1] = a_i$, $\quad t_i[2] = 0$ $\quad$ with release time $R_i = 0$ $\forall i$, $1 \leq i \leq m$.

$t_{m+1}[1] = (2n+1)B$, $\quad t_{m+1}[2] = (2n+1)B$ $\quad$ with release time $= 0$.

$t_{m+2}[1] = (2n + \frac{1}{2})B$, $\quad t_{m+2}[2] = (2n)B$ $\quad$ with release time $(2n+1)B$.

$t_{m+i}[1] = (2n - i + 2 + \frac{1}{2})B$, $\quad t_{m+i}[2] = (2n - i + 2)B$ $\quad$ with release time $R_{m+i} = (2n+1)B + \sum_{j=2}^{i-1}(2n - j + 3)B$ $\forall i$, $3 \leq i \leq 2n$.

$t_{m+2n+1}[1] = 1\frac{1}{2}$, $\quad t_{m+2n+1}[2] = 0$ $\quad$ with release time $((2n+1)(n+2) - 3)B$.

and the schedule length is $((2n+1)(n+2) - 1)B$.

We prove that there exists a 3-partition to the above problem iff there exists a feasible schedule to the above problem.

We illustrate the basic principle of our proof with an example for $n = 2$. See table 2.10

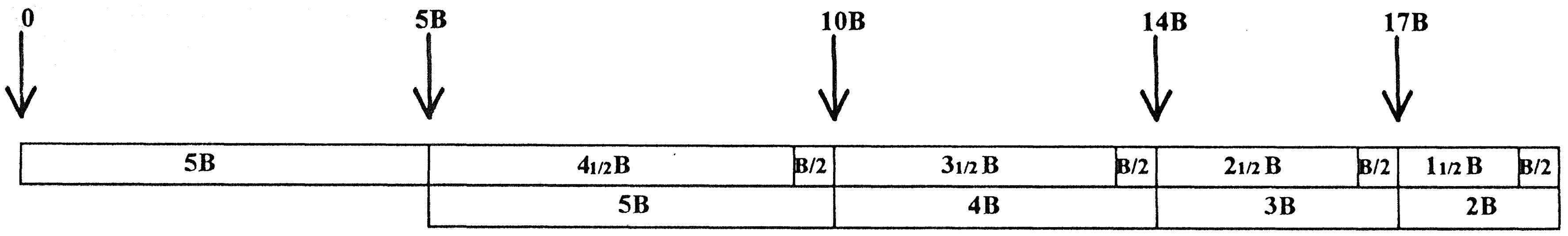
i	$t_i[0]$	$t_i[1]$	$t_i[2]$
1	0	a_1	0
2	0	a_2	0
3	0	a_3	0
4	0	a_4	0
5	0	a_5	0
6	0	a_6	0
7	0	$5B$	$5B$
8	$5B$	$4\frac{1}{2}B$	$4B$
9	$10B$	$3\frac{1}{2}B$	$3B$
10	$14B$	$2\frac{1}{2}B$	$2B$
11	$17B$	$1\frac{1}{2}B$	0

Table 2.10: An example problem for proving NP hardness of a special case

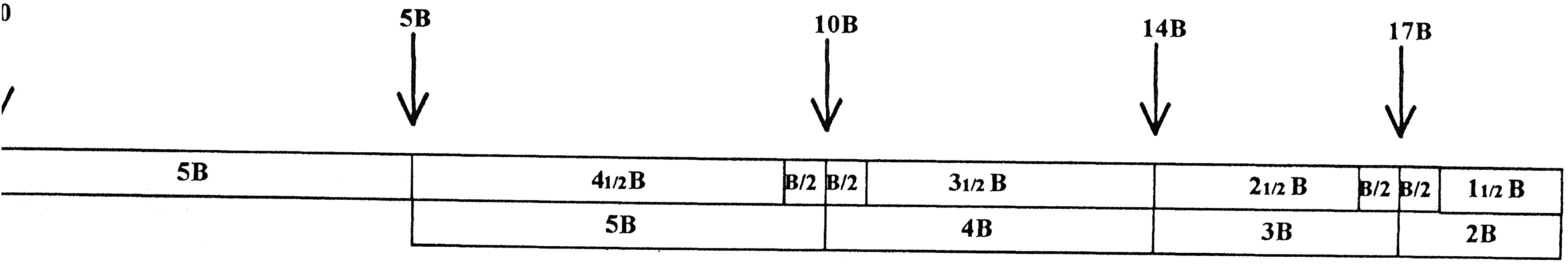
or which schedule length is $19B$.

If there is a 3-partition to the multi set A, then the feasible schedule can be onstructed as shown in the figure 2.1b. Note that $\sum_{i=1}^{11} = 19B$ and also the sum of he unavoidable idle time on second machine and sum of the processing requirement on econd machine is $19B$. Hence in any feasible schedule, job seven has to be scheduled irst and job eleven has to be scheduled last as shown in figure 2.1a. Moreover there annot be any idle time on machine one. The jobs from seven to eleven get processed before $19B$ if they are scheduled as shown in figure 2.1a but they leave idle time slots of maximum length $B/2$ on machine one. Though the sum of idle times on machine one after processing the above jobs is nB, the restriction, $B/4 < a_i < B/2$, on the length of tasks one to six make it impossible for these jobs to be scheduled in the aforementioned idle time slots. Since preemption is not allowed, in any feasible schedule the tasks seven to eleven have to be scheduled as shown in figure 2.1b. This type of arrangement leaves idle time gaps of length B each. Since there is no idle time on machine one and $B/4 < a_i < B/2$, there will be exactly three jobs scheduled in these idle time slots and the sum of these jobs will be B. Therefore if there exists a feasible schedule then 3-partition problem can be constructed and vice versa.

Figures 2.2a and 2.2b show the schedule construction for a general case. If there a 3-partition to the multi set A, then the feasible schedule to the above problem in be constructed as shown in the figure 2.2b. Since sum of the processing times of ie jobs on machine one is equal to $((2n+1)(n+2)-1)B$, there cannot be any idle me on machine one for any feasible schedule. The unavoidable idle time on machine vo, $\min(t_i[0]+t_i[1] \mid t_i[2] \neq 0)$, is $(2n+1)B$ before any processing can take place n machine two. Sum of the unavoidable idle time and processing requirement on iachine two is equal to $((2n+1)(n+2)-1)B$. Hence the job $m+1$ has to be scheduled s the first job and there cannot be any other idle time on machine two in any feasible chedule. Second job is available only after $(2n+1)B$ and has to be processed before $(2n+1)B$ so that second processor remains busy and hence the schedule remains easible. This leaves an idle time of length $\frac{B}{2}$ on machine one. Similar arguments an be given to show that there exist $2n$ unutilized processing time slots of length $B/2$ each on machine one if there is to be no idle time on machine two other than $(2n+1)B$ units of unavoidable idle time in any schedule (see figure 2.2a). The m jobs with nB units of total processing requirement on machine one have to be scheduled in these gaps without leaving any idle time on machine one such that the resulting schedule is feasible. This is possible only if the jobs from $m+1$ to $m+2n+1$ are scheduled as shown in figure 2.2b leaving n gaps of length B units each. Therefore in any feasible schedule the jobs from $m+1$ to $m+2n+1$ are scheduled as shown in the figure 2.2b with the jobs from 1 to m scheduled in n the gaps of length B each. Since $B/4 < a_i < B/2$, there will be exactly three jobs, whose sum is B, scheduled in each idle time slot. Therefore if there is a feasible schedule to the above problem, then there is a 3-partition to the multi set A.

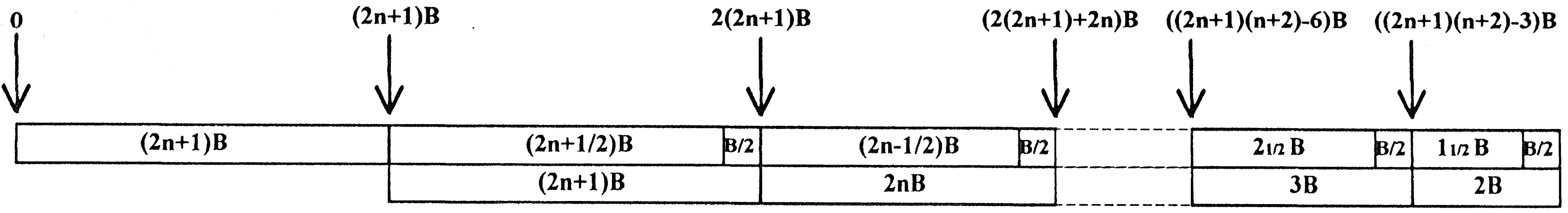


a. Schedule construction for $n = 2$

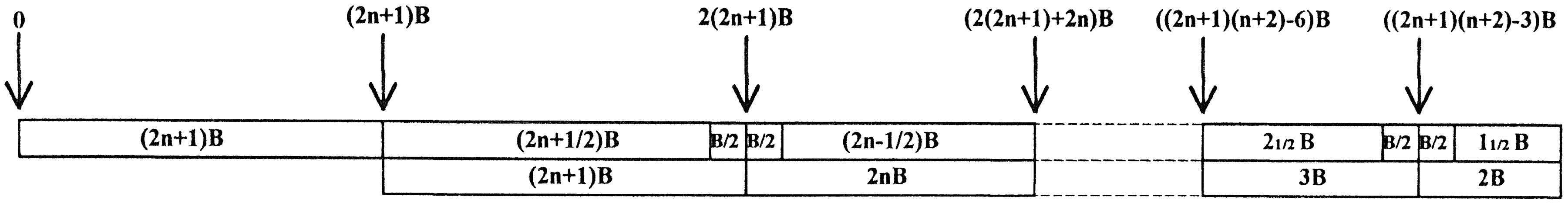


b. Feasible schedule construction for $n = 2$

Figure 2.1: Construction of schedules for $n = 2$



a. Schedule construction for general case



b. Feasible schedule construction for general case

Figure 2.2: Construction of schedules for general case

2.4 Conclusions

In the first section, polynomial time algorithms for three special cases of two machine flow shop problem with total flow time as minimization criteria are presented. First two cases can be solved in $O(n \log n)$ time whereas the third case takes $O(n^2)$ time. A fourth special case is shown to be NP hard. In the second, section polynomial time algorithms for three special cases of two machine flow shop problem with release times and with makespan as optimization criteria are presented. A fourth case is shown to be NP hard.

Chapter 3

EFFECTS OF SPECIAL CASES ON BRANCH AND BOUND ALGORITHM TO SOLVE GENERAL FLOW SHOP SCHEDULING PROBLEM

3.1 Introduction

In the scheduling literature, it has been proved that for make span criterion, the two machine flow shop problem with release times and three machine flow shop problem without release times are NP hard. But there are some special cases of the above problems which can be solved in polynomial time. When a problem does not belong to any of the special cases, it is solved using either explicit or implicit enumeration techniques to get an optimum solution. Normally the search is curtailed using lower

›ounding techniques or using dominance rules.

The dominance rules are also known as elimination methods in literature. The ɔasic principle used in this type of solution method is that for a given partial sequence, ;wo jobs i and j are checked to see whether one of them dominates other. If the dominance is established, then the dominated sequences are eliminated from final consideration. Similar procedure is repeated with further partial sequences until all the remaining partial sequences are undominated which are explicitly enumerated for an optimal sequences.

Ignal and Schrage [64] presented first a set of machine based bounds for solving flow shop problem with make span criteria. Bestwick and Hastings [13] have developed job based bounds for the same problem. Lagweg [74] has presented a general theory of bounding for this problem.

Dudek and Teuton [36] were the first ones to report dominance conditions through combinatorial analysis. Their algorithm begins with a partial sequence σ, which is a set of already scheduled jobs. Two jobs, i and j are compared by $(m - 1)$ conditions. If all the conditions are met, then job j is eliminated from consideration. This algorithm was originally thought to be a general one to get an optimal sequence but a counterexample formulated by Karush [68] proved otherwise. Later Smith and Dudek [105] eliminated the drawbacks of this algorithm and also gave a proof of sequence optimality.

For the three machine flow shop problem, Gupta and Reddi [56] developed new dominance conditions which eliminated an important drawback of the existing dominance conditions [55, 78, 112] that the jobs should have their least processing time on either first or third machine. Szwarc in [112] proved that the elimination methods developed by him in [111] are optimal and outlined a method to construct a general counterexample to any procedure that claims to remove more sequences than this optimal method. He also gave a counterexample to show that elimination methods given y Bagga and Chakravarti [9] eliminate all optimal sequences from consideration and hus result in suboptimal solutions. Through another work [113], Szwarc examines he mathematical structure of a dominance condition given by Gupta and Reddi [56] or the three machine flow shop problem. He showed that the condition developed n [56] may eliminate fewer sequences than the conditions developed in [55, 78, 112]. Then he gives a sequential procedure to combine the two conditions in order to achieve etter results.

Baker [11] by his experimental study concluded that the algorithms based on dominance criteria are inferior to algorithms based on branch and bound methods. He suggests use of a composite approach which uses branch and bound technique on the dominant set of sequences.

The dominance checking process is time consuming and may generate significant number of dominant schedules. In this chapter we study the effectiveness of special cases in deciding the dominance of one partial sequence over others. We also present an effective method of using these dominance rules to curtail the enumeration in *B&B* algorithms. Then, we study the effects of dominance rules on computational aspects of the branch and bound technique to solve flow shop scheduling problems. Since the emphasis is on the usefulness of the special cases, we restrict ourselves to the problems that are known to have polynomially solvable special cases.

The *organization* of the chapter is as follows. In section 3.2 we study the two machine flow shop problem with release times and in section 3.3 the three machine flow shop problem without release times. Each of these sections is supplemented with computational results. Section 3.4 concludes the chapter.

3.1.1 Notations

The following notations is used in this chapter. It is similar to the notation given in the previous section 2.1.1.

$t_i[0]$ be the release time of the job i.

$t_i[1]$ be the processing time of the job i on the first machine.

$t_i[2]$ be the processing time of the job i on the second machine.

$t_i[3]$ be the processing time of the job i on the third machine.

σ be the partial sequence constructed at any stage.

σ' be the set of jobs not contained in σ.

C_i be the completion time of the job i on the first machine.

3.2 Two Machine Flow Shop Problem with Release Times

The characteristics of the problem are as follows: There are n jobs to be processed on two machines. All jobs are to be processed in the same sequence. All the jobs are not available at time $t = 0$, and the arrival times of all the jobs are known. The minimization criteria is schedule length (make span). The general problem in this category is known to be NP hard [24]. Nevertheless, there are special cases of this problem which can be solved in polynomial time (see chapter 2). In this section we give a lemma and a theorem which are essential for studying the effects of the special cases on the general problem.

Notations :

The following notation is specific to this section:

K_i = Idle time on the first machine before the processing of the i^{th} job.

K_i = Idle time on the second machine before the processing of the i^{th} job.

Lemma 2 *In any schedule S, there exists a job j such that, after its completion on machine one, all the jobs that are not processed till then are available for processing without further delay.*

Proof : Trivial.

The consequence of the above lemma is that a sequence S can be broken into two partial sequences S_1 and S_2 such that when last job in S_1 is completed all the jobs in S_2 would be available for processing.

Theorem 5 *For the set of the jobs which are available after processing of the job j such that $C_j \geq t_{\max}[0]$, the Johnson's sequence is the optimum sequence.*

Proof: From previous lemma it can be inferred that a schedule S can be broken into two partial sequences S_1 and S_2 such that, S_1 is the set of jobs which are processed before C_j and S_2 is the set of the jobs which can be processed after C_j.
$S_1 \bigcup S_2 = S$ and $S_1 \bigcap S_2 = \emptyset$.

Consider the inequality to minimize idle time on the second machine, given in section 2.3.

$$\max(H_j + K_{u|u\leq j}, H_{j+1} + K_{u|u\leq j+1}) < \max(H'_j + K'_{u|u\leq j}, H'_{j+1} + K'_{u|u\leq j+1})$$

Where

$$H_j = \sum_{i=1}^{j} t_i[1] - \sum_{i=1}^{j-1} t_i[2],$$

and

$$K_u = t_u[0] - \sum_{i=1}^{u-1} t_i[1]$$

Consider the expressions for X_i and Y_i

$$X_i = \max(t_i[0] - (\sum_{j=1}^{i-1} t_j[1] + \sum_{j=1}^{i-1} X_j), 0) \tag{3.1}$$

$$Y_i = \max(\sum_{j=1}^{i} t_j[1] + \sum_{j=1}^{i} X_j - (\sum_{j=1}^{i-1} t_j[2] + \sum_{j=1}^{i-1} Y_j), 0) \tag{3.2}$$

In equation 3.1, the term $\sum_{j=1}^{i-1} t_j[1]+\sum_{j=1}^{i-1} X_j$ is the completion time of the $(i-1)^{th}$job. When a job r, such that $C_r \geq t_{\max}[0]$ is processed, for the jobs in $\sigma', X_{i|i\in\sigma'} = 0$. Therefore, $\sum_{i=1}^{n} X_i = \sum_{i=1}^{r} X_i$, where r is the last job in σ.

Now consider the cumulative delay on the second machine, $\sum_{i=1}^{n} Y_i$.

$$\begin{aligned} \sum_{i=1}^{n} Y_i &= \max(\sum_{i=1}^{n} t_i[1] - \sum_{i=1}^{n-1} t_i[2] + \sum_{i=1}^{n} X_i, \sum_{i=1}^{n-1} Y_i) \\ &= \max_{1\leq v\leq n}(H_v + K_{u|u\leq v}) \end{aligned} \tag{3.3}$$

Where

$$H_v = \sum_{i=1}^{v} t_i[1] - \sum_{i=1}^{v-1} t_i[2],$$

and

$$K_u = t_u[0] - \sum_{i=1}^{u-1} t_i[1]$$

Let S be the sequence with some arrangement of jobs in S_1 and S_2 and S' be the sequence in which the jobs in S_1' are in the same order as that of S_1 and the jobs j and $j+1$ in S_2 are interchanged. The schedule S will be better than S' if $\sum_{i=1}^{n} Y_i < \sum_{i=1}^{n} Y_i'$

, where $\sum_{i=1}^{n} Y_i'$ is the total idle time on the second machine for the schedule S'. The only terms that undergo change in $\sum_{i=1}^{n} Y_i$ and $\sum_{i=1}^{n} Y_i'$ are the ones associated with j and $j+1$.
Therefore S will be better than S' if

$$\max(H_j + K_{u|u\le j}, H_{j+1} + K_{u|u\le j+1}) < \max(H_j' + K'_{u|u\le j}, H'_{j+1} + K'_{u|u\le j+1}) \quad (3.4)$$

Now for a given S_1 and S_2 with the interchanged jobs, the term K_u will be constant and equal to K_r where r is the cardinality of S_1. Therefore, the inequality 3.4 can be written as

$$\max(H_j + K_r, H_{j+1} + K_r) < \max(H_j' + K_r, H'_{j+1} + K_r)$$

After subtracting K_r, substituting expressions for H_j and H_{j+1}, and subtracting $\sum_{i=1}^{j+1} t_i[1] - \sum_{i=1}^{j-1} t_i[2]$ from the above inequality, we get

$$\max(-t_j[2], -t_{j+1}[1]) < \max(-t_{j+1}[2], -t_j[1])$$

$$\Longrightarrow \min(t_j[1], t_{j+1}[2]) < \min(t_{j+1}[1], t_j[2]).$$

This inequality is same as the one derived by Johnson for the two machine problem. Therefore it can be inferred that for a given S_1, the Johnson's algorithm gives optimal sequence for S_2 i.e., Johnson's sequence for S_2 dominates all other sequences for S_2.

Now, we can see the effect of the theorem 5 on the branch and bound algorithm. Let us consider a node x where the above situation occurs. Let n_1 be the cardinality of the set S_1 and n_2 be the cardinality of the set S_2. It follows from theorem 5 that we need not branch out from the node x further because for the set S_2 , Johnson's sequence gives the optimum. If the result of the above theorem is not used, then in the worst case, we have to branch $(n_2 - 1)!$ times from that node. This worst case branching can be avoided only at the expense of checking for the occurrence of the condition stated in the lemma. The cost of the checking includes computing C_i and a comparison of C_i and $t_{\max}[0]$ resulting in $(O(1))$ time.

We can generalize the above theorem 5 and obtain some interesting results. Let ready times of the jobs be such that, by the time any one of the jobs is processed, all other jobs are available for processing. Then there are n sets of type S_1. For a given S_1, we can apply Johnson's algorithm to S_2 to get an optimal sequence for that S_1. Since there are n sets of type S_1, we get n different sequences which dominate all other sequences. Hence, the best out of these n sequences is the optimal sequence. This result coincides with the result given in the previous chapter. This result can be further extended to cover the following situation : when any r jobs have been processed, the rest of the jobs will be available for processing without further delay. In this case, there will be nC_r sets of type S_1. Hence there will be nC_r number of sequences which dominate all other solutions. To get a single solution, the complexity is $O(n \log n)$. Hence to get an optimal sequence, the total effort required will be $O(nC_r n \log n)$.

3.2.1 Computational Results

Computational tests were conducted to observe the effectiveness of the dominance rule developed earlier in this section. For this purpose a branch and bound algorithm based on machine based bounds is used. It uses best incumbent first strategy to fathom the search tree. Whenever a candidate node is selected for fathoming, applicability of the dominance rule is checked. The results obtained by this procedure (BBD) are compared with those given by branch and bound procedure (BB) that ignores the dominance rule.

The tests were conducted on a HP 9000/850 supermini computer. For all the

Category	Distribution, U $\sim$ (1,α)
1	Release Times $\sim$ (1,100)
2	Interarrival Times $\sim$ (1,10)
3	Interarrival Times $\sim$ (1,30)
4	Interarrival Times $\sim$ (1,50)
5	Interarrival Times $\sim$ (1,70)
1,2,3,4 and 5	Processing Times $\sim$ (1,100)

Table 3.1: Inter arrival times/release times, and processing times distributions

problems, processing times, and release times/inter arrival times random numbers generated from the uniform distribution with the range 1 to α. In category 1 of the problems, release times are generated directly while in other categories they are obtained through the generation of inter arrival times. Values of α for various categories of problems are given in table 3.2.1. The problem sizes (number of jobs in a job set) in each category are varied from ten to fifty in steps of five. For each problem size, ten problems are solved. The metrics used for comparison are average number of nodes generated and average CPU time.

The results of the computational tests are tabulated in the tables 3.2 to 3.6. The same results are graphically depicted in the figures 3.1 to 3.5.

It can be seen from the results that the dominance rule proposed in this work is most effective for category 1 problems. For other categories the effectiveness of the dominance rule decreases as α increases. In fact as α reaches 70, branch and bound algorithm seems to perform better without the dominance rule. The reason for the above behavior is that in a B&B search tree, as α increases, the depth at which the condition of the dominance rule is satisfied increases. This causes a decrease in the savings in terms of number of nodes generated and thus causes an increase in CPU time.

	No. Nodes		CPU Time	
Size	BB	BBD	BB	BBD
10	67.18	24.91	5.64	4.36
15	122.45	32.45	6.82	4.73
20	211.00	45.00	10.11	6.56
25	318.89	58.89	17.12	9.73
30	482.00	74.56	27.44	11.78
35	642.82	92.82	36.18	16.36
40	834.09	136.91	49.55	22.82
45	1117.90	132.21	81.62	27.09
50	1275.18	142.27	87.64	36.45
55	1551.91	152.09	115.73	27.45

Table 3.2: Results for category 1 problems; Release time $\sim$ $(1,\alpha)$

	No. Nodes		CPU Time	
Size	BB	BBD	BB	BBD
10	62.20	19.70	4.60	3.70
15	121.00	31.50	7.00	5.80
20	296.30	87.00	15.20	10.60
25	333.60	58.10	19.40	11.20
30	472.40	100.20	26.50	18.60
35	631.80	111.20	39.00	27.20
40	822.89	141.22	54.56	32.22
45	1041.67	168.89	73.78	42.00
50	1305.50	230.40	103.70	56.20
55	1562.50	304.50	133.00	94.00

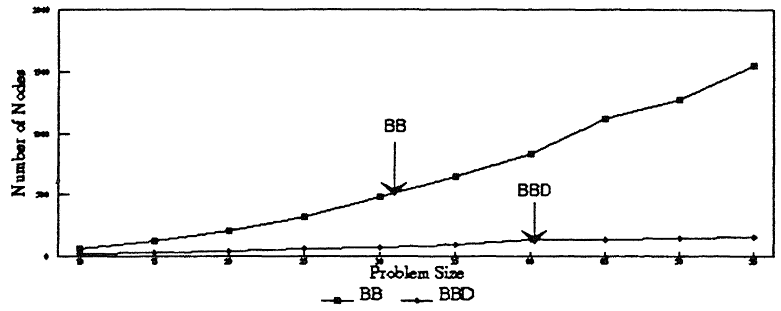
Table 3.3: Results for category 2 problems; Inter arrival times $\sim$ $(1,\alpha)$

	No. Nodes		CPU Time	
Size	BB	BBD	BB	BBD
10	61.8	25.8	25.8	4.8
15	121.9	57.2	57.2	5.6
20	211.6	94.3	94.3	9.3
25	343.1	159.3	159.3	12.5
30	468.5	241	241	22.9
35	641.3	308	308	31.4
40	840.3	395.1	395.1	54.5
45	1076.7	563	563	63.4
50	1312.9	579.3	579.3	85.9
55	1559.6	749.1	749.1	118.3

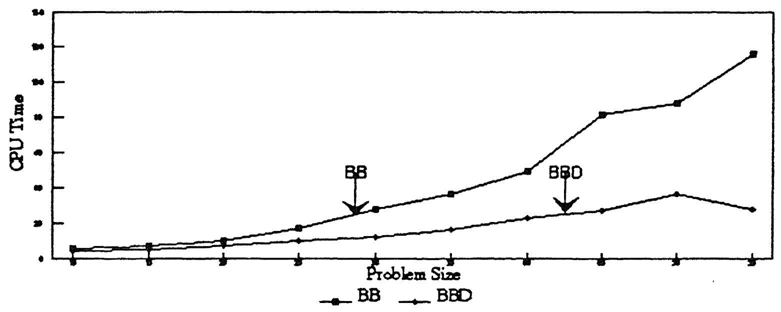
Table 3.4: Results for category 3 problems; Inter arrival times $\sim$ (1,α)

	No. Nodes		CPU Time	
Size	BB	BBD	BB	BBD
10	264.9	40.6	20.1	4.7
15	157.9	127.1	8.1	9.2
20	227.8	155.2	11.1	10.3
25	400.9	296.2	20	19.4
30	499	349.8	26.7	25.1
35	648.5	450	40.7	36
40	858.7	630	55.7	62.4
45	1077.8	745.1	75	65.8
50	1399.3	1039.8	104.4	94.8
55	1622.2	1145.4	124.3	121.2

Table 3.5: Results for category 4 problems; Inter arrival times $\sim$ (1,α)

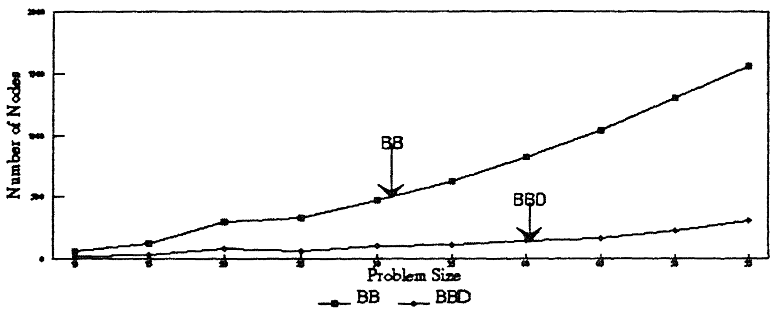


(a)

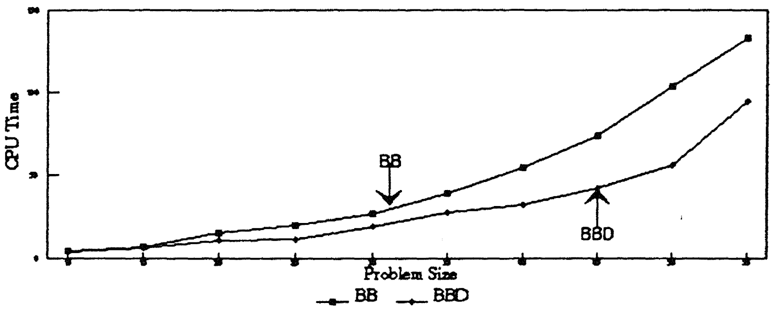


(b)

Figure 3.1: Graphical representation of results for category 1 problems

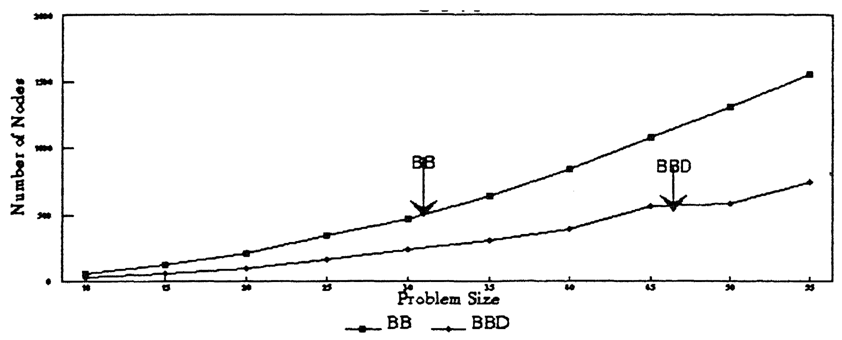


(a)

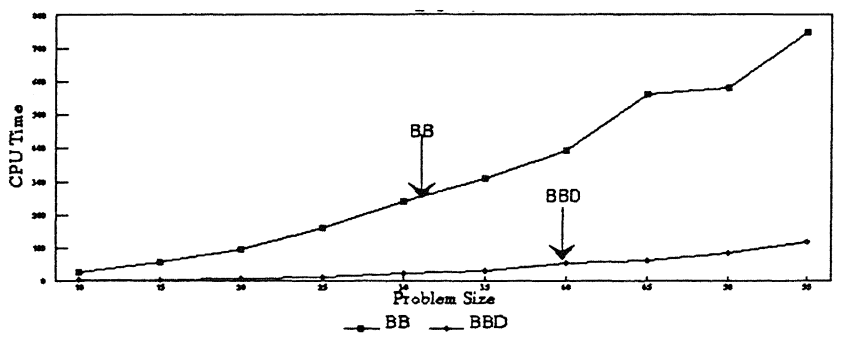


(b)

Figure 3.2: Graphical representation of results for category 2 problems

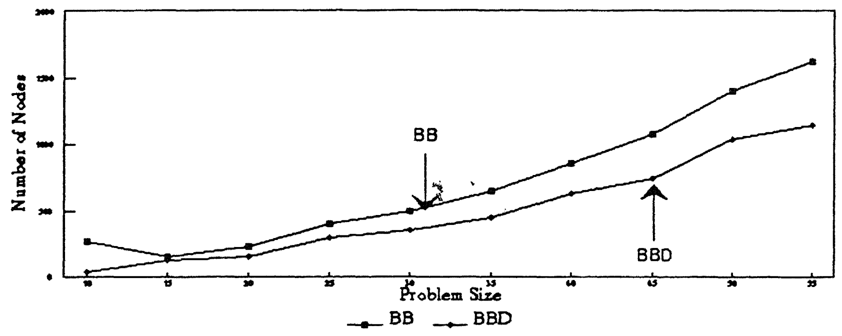


(a)

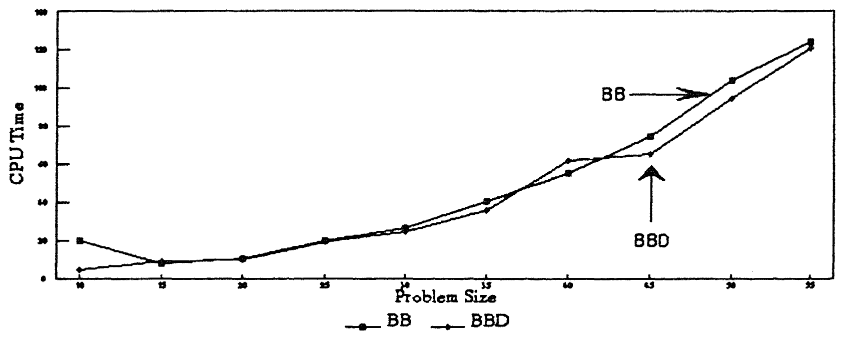


(b)

Figure 3.3: Graphical representation of results for category 3 problems

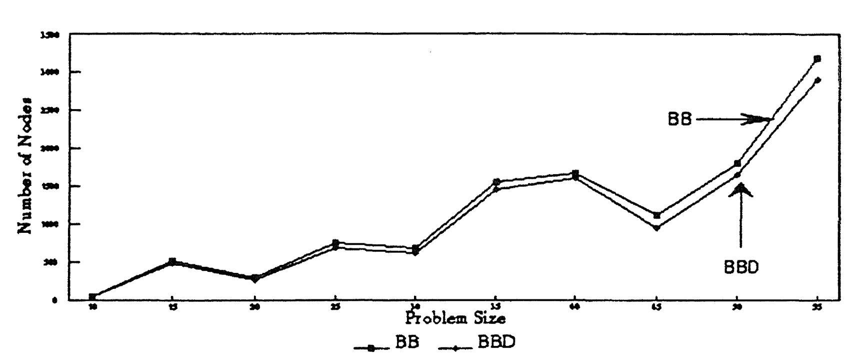


(a)

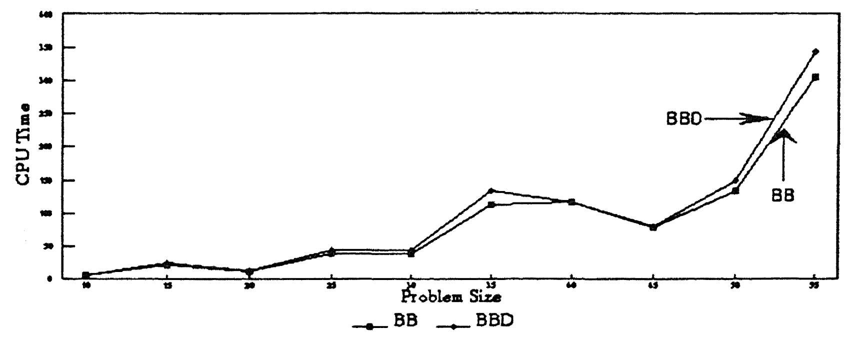


(b)

Figure 3.4: Graphical representation of results for category 4 problems



(a)



(b)

Figure 3.5: Graphical representation of results for category 5 problems

	No. Nodes		CPU Time	
Size	BB	BBD	BB	BBD
10	58.9	45.3	5.3	5.3
15	521.2	492.6	21.8	25.2
20	304.8	262.6	12.7	13.2
25	751.2	684.4	39.2	43.5
30	683	616.7	38.8	44.5
35	1561.5	1457	113.6	133.1
40	1679	1605.5	116.2	117
45	1116.8	951.6	79.3	80.8
50	1808.3	1649.4	134.8	148.5
55	3187.2	2905.5	305.3	343.2

Table 3.6: Results for category 5 problems; Inter arrival times $\sim$ (1,α)

3.3 Three Machine Flow Shop Problem without Release Times

In this section we discuss the three machine flow shop problem. In section 3.1 we deal with the special case considered by Arthanary and Mukhopadhay [5], $\min t_i[2] \geq \max t_i[1]$ and in section 3.2 we deal with the case considered by Johnson [67], $\min t_i[1] \geq \max t_i[2]$.

Notations: specific to this three machine flow shop problem is given below:
X_i = Idle time on the second machine before the processing of the i^{th} job
Y_i = Idle time on the third machine before the processing of the i^{th} job.

3.3.1 Dominance Rule for Special Case, $\min t_i[2] \geq \max t_i[1]$

Lemma 3 *Any job set S can be broken into two partial schedules S_1 and S_2, where the set S_1 contains any ordering of jobs, and set S_2 contains the jobs that obey the special case*

Proof: Proof of the above lemma is trivial since we allow the set S_2 to contain zero elements.

Theorem 6 *If the jobs in the set S_2 obey the rule $\min t_i[2] \geq \max t_i[1]$, then for a given S_1, the Johnson's sequence gives the optimal seqûence for the jobs in S_2.*

Proof : Proof of this theorem is similar to that of the previous theorem 5 but for the changes in expressions of X_i and Y_i. Consider the expressions given for X_i and Y_i in Johnson's paper [67]:

$$X_i = \max(\sum_{j=1}^{i} t_j[1] - \sum_{j=1}^{i-1} t_j[2] - \sum_{j=1}^{i-1} X_j, 0)$$

$$Y_i = \max((\sum_{j=1}^{i} t_j[2] + \sum_{j=1}^{i} X_j) - (\sum_{j=1}^{i-1} t_j[3] + \sum_{j=1}^{i-1} Y^j), 0)$$

and

$$\sum_{i=1}^{n} Y_i = \max(\sum_{j=1}^{n} t_j[2] + \sum_{j=1}^{n} X_j - \sum_{j=1}^{n-1} t_j[3], \sum_{j=1}^{n-1} Y_i)$$

$$\sum_{i=1}^{n} Y_i = \max_{1 \leq u \leq v \leq n} (H_v + \max K_u)$$

where

$$H_v = \sum_{i=1}^{v} t_i[2] - \sum_{i=1}^{v-1} t_i[3]$$

$$K_u = \sum_{i=1}^{u} t_i[1] - \sum_{i=1}^{u-1} t_i[2]$$

Let S be a sequence broken into S_1 and S_2 with any arrangement of jobs in S_1 and with S_2 containing jobs which obey the special case constraint. Let S' be the altered sequence with S_1' having same sequence as S_1 and S_2' having jobs j and $j+1$ interchanged from S_2. The sequence S will be better than S' if

$\sum_{i=1}^{n} Y_i < \sum_{i=1}^{n} Y_i'$, where $\sum_{i=1}^{n} Y_i'$ is the sum of the idle times on the third machine with sequence S'.

Let r be the last job in S_1.

For $r < v \leq n$,

$$\max_{1 \leq u \leq v} K_u = \max_{1 \leq u \leq r} K_u$$

Since the last job in S_1 is r and the jobs in S_2 follow the special case constraint, $\min t_i[2] \geq \max t_i[1]$.

For the interchange of the jobs in S_2, the term K_u will remain constant. Therefore the differences in the sum of idle times for both the sequences will arise only because of the j and $j+1$ terms.

The inequality

$\sum_{i=1}^{n} Y_i \leq \sum_{i=1}^{n} Y_i'$

can be written as

$$\max(H_j, H_{j+1}) < \max(H_j', H_{j+1}')$$

Substituting expressions for H_j, H_{j+1}, H_j' and H_{j+1}', we get

$$\begin{gathered} \max(\sum_{i=1}^{j} t_i[2] - \sum_{i=1}^{j-1} t_i[3], \sum_{i=1}^{j+1} t_i[2] - \sum_{i=1}^{j} t_i[3]) \;\; < \\ \max(\sum_{i=1}^{j} t_i'[2] - \sum_{i=1}^{j-1} t_i'[3], \sum_{i=1}^{j+1} t_i'[2] - \sum_{i=1}^{j} t_i'[3] \end{gathered} \tag{3.5}$$

Subtracting $(\sum_{i=1}^{j+1} t_i[2] - \sum_{i=1}^{j-1} t_i[3])$ from this inequality, we get

$$\max(-t_{j+1}[2], -t_j[3]) < \max(-t_j[2], -t_{j+1}[3])$$

$$\Longrightarrow \min(t_j[2], t_{j+1}[3]) < \min(t_{j+1}[2] + t_j[3]) \tag{3.6}$$

The inequality 3.6 is same as the one derived for Johnson's two machine problem with $t_i[2]$ and $t_i[3]$ as the processing times. Note that the above rule gives optimal partial sequence for S_2 only for a given sequence of jobs in S_1.

The effect of the above theorem 6 on branch and bound algorithm is as follows. Let x be the node at which the above mentioned situation occurs, n_1 be the cardinality of the set S_1, and n_2 be the cardinality of the set S_2. From the theorem 6 it is clear that the node x need not be explored any further. The saving in the computation is same as the one reported in previous section. But the cost of checking for the occurrence special case condition $O(n)$ is more expensive than that in the previous case $O(n)$. This calls for the trade off between the cost of checking every node and the benefit of saving the branching at every eligible node.

It can be easily seen that the theorem 6 can be generalized only for one case i.e., when all the jobs in job set satisfy the special case condition. Even if all the jobs satisfy the above condition, the set S_1 should contain at least one job. This directly follows from Lemma 2 which implies that there should be at least one job which can divide a sequence into two partial sequences S_1 and S_2, and that job is included in set S_1. In the above generalization any single job can divide the initial sequence into two sets. Therefore there are n distinct sets of type S_1 and S_2. For each given S_1 optimal arrangement of jobs in S_2 can be found using Johnson's rule. Then, the best out of these n sequences is the optimal sequence. This result agrees with that of the Arthanary's [5].

3.3.2 Dominance Rule for Special Case, $\min t_i[1] \geq \max t_i[2]$

In this section we consider Johnson's special case for three machine scheduling problem i.e., $\min t_i[1] \geq \max t_i[2]$.

Theorem 7 *If the jobs in set S_1 obey the rule* $\min t_i[1] \geq \max t_i[2]$, *the Johnson's sequence gives optimal arrangement of the jobs in S_1 for a given arrangement of jobs in S_2.*

Proof : Proof of this theorem follows directly from Johnson's proof [67].
Let r be the last job in the set S_1.

Let S be the sequence with partial sequences S_1 and S_2 and S' be the sequence with partial sequences S'_1 and S'_2 such that S'_2 contains same sequence of jobs as in S_2 but S'_1 has the jobs j and $j+1$ interchanged as compared with S_1 for any $j \leq r-1$. Consider the inequality given in Johnson's paper [67] for schedule S to be better than S':

$$\max(H_j + K_{u|u \leq r}, H_{j+1} + K_{u \leq j+1}) < \max(H'_j + K'_{u|u \leq j}, H'_{j+1} + K'_{u|u \leq j+1} \qquad (3.7)$$

where the terms $H_j, H_{j+1}, H'_j, H'_{j+1}, K_u$, and K'_u are as defined in the previous sub-section.
Since $K_{u|1 \leq u \leq i} = K_i$ for $i \leq r-1$, the inequality 3.7 can be written as

$$\max(H_j + K_j, H_{j+1} + K_{j+1}) < \max(H'_j + K'_j, H'_{j+1} + K'_{j+1}) \qquad (3.8)$$

By substituting $H_j = \sum_{i=1}^{j} t_i[2] - \sum_{i=1}^{j-1} t_i[3]$ and $K_j = \sum_{i=1}^{j} t_i[1] - \sum_{i=1}^{j-1} t_i[2]$ in 3.8, we get

$$\max\begin{pmatrix} \sum_{i=1}^{j} t_i[2] - \sum_{i=1}^{j-1} t_i[3] + \sum_{i=1}^{j} t_i[1] - \sum_{i=1}^{j-1} t_i[2], \\ \sum_{i=1}^{j+1} t_i[2] - \sum_{i=1}^{j} t_i[3] + \sum_{i=1}^{j+1} t_i[1] - \sum_{i=1}^{j} t_i[2] \end{pmatrix} < \qquad (3.9)$$

$$\max\begin{pmatrix} \sum_{i=1}^{j} t'_i[2] - \sum_{i=1}^{j-1} t'_i[3] + \sum_{i=1}^{j} t'_i[1] - \sum_{i=1}^{j-1} t'_i[2], \\ \sum_{i=1}^{j+1} t'_i[2] - \sum_{i=1}^{j} t'_i[3] + \sum_{i=1}^{j+1} t'_i[1] - \sum_{i=1}^{j} t'_i[2] \end{pmatrix}$$

By subtracting $\sum_{i=1}^{j+1} t_i[2] - \sum_{i=1}^{j-1} t_i[3] + \sum_{i=1}^{j+1} t_i[1] - \sum_{i=1}^{j-1} t_i[2]$ from 3.9, we get

$$\max(-t_{j+1}[2] - t_{j+1}[1], -t_j[3] - t_j[2]) < \max(-t_j[2] - t_j[1], -t_{j+1}[3] - t_{j+1}[2])$$

$$\Longrightarrow \min(t_j[1] + t_j[2], t_{j+1}[2] + t_{j+1}[3]) < \min(t_{j+1}[1] + t_{j+1}[2], t_j[2] + t_j[3])$$

The above rule is same as Johnson's rule for $1 \leq j \leq r-1$ i.e., Johnson's rule gives optimal arrangement of jobs for the job set in S_1 irrespective of the arrangement of the jobs in set S_2. Hence the theorem.

The result of the above theorem 7 can be utilized when the *B&B* algorithm constructs a schedule from back to front. The costs and benefits associated with this rule are same as that of the previous case.

It is easy to extend the above result when all the jobs follow the rule
$\min t_i[1] \geq \max t_i[2]$. When this situation occurs, the set S_2 will be empty and the problem becomes Johnson's problem [67].

3.3.3 Computational Results

Computational tests were conducted to observe the performance of the dominance rule developed in subsection 3.3.1 relative to the dominance rules due to Gupta and Reddi [56] and Szwarc [111] in curtailing the number of nodes generated by branch and bound algorithm. With conditions due to [56] and [111] a node's children are checked for dominance by one of others while with our dominance rule only the node itself is checked. The computational results are also compared with a branch and bound algorithm which does not use any dominance rules. Tests were also conducted for verifying the effectiveness of our dominance rule (section 3.3.1) while solving biased problems with *B&B*. Biased problems are the problems that have a fixed percentage (say β) of the jobs that obey special case constraints, in this case $\min t_i[2] \leq \max t_i[1]$. All the *B&B* methods use simple machine based bounds.

The tests were conducted on a HP 9000/850 supermini computer. Processing times of the jobs are random numbers that are uniformly distributed between one and hundred. In biased problems, the jobs that obey the special case conditions have the processing times on the first machine uniformly distributed between one and fifty and the processing times on the second machine uniformly distributed between fifty and hundred. For comparison of different dominance rules, problems of two sizes, ten and thirty, are used. For biased problems, the problem size is varied from ten to fifty in steps of ten and β (percentage of special case jobs) is varied from thirty to ninety in steps of twenty. For each problem size, ten problems are solved. The metrics used for comparison are average number of nodes generated and CPU time.

The results of computational testing for different dominance rules are tabulated in the table 3.7. Abbreviations used in table 3.7 denote the following:

BB	Branch and Bound
ReddiBB	Branch and bound with dominance rules developed in [56]
SzwrcBB	Branch and bound with dominance rules developed in [111]
SivaBB	Branch and bound with dominance rules developed in section 3.3.1

The relative performance of various rules are as follows:

Number of nodes: $SzwrcBB < ReddiBB < SivaBB < BB$.

CPU Time : $BB < SivaBB < SzwrcBB <<< ReddiBB$.

It can be seen from the results that the dominance rule of Szwarc [111] is the most effective of all in curtailing the number of nodes generated. But the cost (CPU time) of checking these conditions is much more than the time advantage gained by curtailing the number of nodes, thus resulting in deteriorated time performance. The savings in number of nodes generated due to the dominance rule of Gupta and

	Problem Size = 10		Problem Size = 30	
Dom. Rule	No. Nodes	CPU time	No. Nodes	CPU time
BB	145.2	8.7	484.1	32.1
ReddiBB	131.4	23.4	478.6	1496.4
SzwrcBB	100.1	10.5	280.3	169.5
SivaBB	141.2	9.9	483.6	36

Table 3.7: Comparative results for different dominance rules

Reddi [56] are very little and hence the time is very high. With SivaBB the saving in number of nodes are little and CPU time required is marginally higher than BB but is lower than that of the other methods. This is because in a random problem, it is unlikely that a significant portion of the job set obeys a special case constraint. Since the cost of a check for this constraint is much less than that for the other conditions, and the number of times this condition is checked is less, CPU time required for SivaBB is less than that for the others. Hence, in case of random problems, use of any of these dominance rules in B&B does not seem to offer any advantage over simple B&B algorithm.

The results of computational tests for biased problems are given in the tables from 3.8 to 3.12. The results pertaining to the behavior of BB and SivaBB for a fixed problem size of thirty and for varying percentage of special case jobs are extracted into the table 3.13. These extracted results are graphically depicted in the figure 3.6. From these results it is obvious that the performance of both BB and SivaBB gets better as the β increases. But improvement in case of SivaBB is enormous. For number of nodes generated, SivaBB is always better than BB but not so for computational time. Still, for $\beta > 50$, our dominance rule (SivaBB) provides significant advantage over the ordinary BB.

	No. Nodes		CPU Time	
Size	BB	SivaBB	BB	SivaBB
10	105.4	104.6	5.7	7.4
20	275.1	231.4	15.3	14.1
30	636.5	633.1	47.6	55.6
40	907.7	901.1	74.1	85.7
50	1331.2	1324.6	135.1	149.5

Table 3.8: Comparative results for $\beta = 10$.

	No. Nodes		CPU Time	
Size	BB	SivaBB	BB	SivaBB
10	70.1	63.7	5.1	5.9
20	227.5	217.8	14.1	14.1
30	474.4	453.3	33.4	35.5
40	821.4	791.6	68.4	71.6
50	1276	1220.5	121.2	128.3

Table 3.9: Comparative results for $\beta = 30$.

	No. Nodes		CPU Time	
Size	BB	SivaBB	BB	SivaBB
10	202.6	191	12.2	13.6
20	217.5	187.8	13.1	12.2
30	469.7	381.6	34.1	30.9
40	829.6	695.7	69.1	64
50	1281	1078.1	127	112.2

Table 3.10: Comparative results for $\beta = 50$.

	No. Nodes		CPU Time	
Size	BB	SivaBB	BB	SivaBB
10	59.7	37.1	5.1	4.6
20	222.5	154.2	13.7	10.9
30	467.4	286.6	32.9	25.5
40	823.8	489	70	46.2
50	1275.2	686.2	123.6	78.8

Table 3.11: Comparative results for $\beta = 70$.

	No. Nodes		CPU Time	
Size	BB	SivaBB	BB	SivaBB
10	56.4	16.8	5.4	4.6
20	210.2	75.4	13.6	8.5
30	465.9	108.5	32.8	12.2
40	821.7	260.2	68.3	28.8
50	1278.9	302.2	123.4	39.8

Table 3.12: Comparative results for $\beta = 90$.

	No. Nodes		CPU Time	
Percent	BB	SivaBB	BB	SivaBB
10	636.5	633.1	47.6	55.6
30	474.4	453.3	33.4	35.5
50	469.7	381.6	34.1	30.9
70	467.4	286.6	32.9	25.5
90	465.9	108.5	32.8	12.2

Table 3.13: Relative performance of SivaBB for problem size of 30 and for varying values of β.